

Ubuntu Linux从入门到精通 (版本9)

邢国庆 仇鹏涛 陈极珺 编著

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

本书首先介绍Ubuntu Linux系统的硬盘安装过程与GNOME桌面环境,然后从最基本的命令行与文件系统基础知识及操作入手,由浅入深,逐步阐述Linux系统的基本概念与原理。在此基础上,对Linux系统的Shell编程、用户管理、进程管理、磁盘空间管理、软件管理、文件系统管理、系统启动过程、作业调度与系统日志,以及TCP/IP网络管理与应用等内容进行深入的讨论。

本书内容丰富,语言流畅,涵盖了Linux系统的主要课题,可以用做大专院校操作系统专业师生的教学参考书,也可作为IT行业人员学习Ubuntu Linux系统的工具书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

Ubuntu Linux从入门到精通(版本9)/邢国庆,仇鹏涛,陈极珺编著.—北京:电子工业出版社,2010.7
ISBN978-7-121-11205-8

I. ①U… II. ①邢… ②仇… ③陈… III. ①Linux操作系统 IV. ①TP316.89

中国版本图书馆CIP数据核字(2010)第121638号

责任编辑:李红玉

文字编辑:易 昆

印 刷:北京天竺颖华印刷厂

装 订:三河市鑫金马印装有限公司

出版发行:电子工业出版社

北京市海淀区万寿路173信箱 邮编:100036

北京市海淀区翠微东里甲2号 邮编:100036

开 本:787×1092 1/16 印张:34.5 字数:883千字

印 次:2010年7月第1次印刷

定 价:64.00元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010)88254888。

质量投诉请发邮件至zlts@phei.com.cn,盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线:(010)88258888。

前 言

Linux系统自1991年诞生以来，吸引了大批人士投身其研发队伍中，以至于Linux流派纷呈，不同品牌的Linux系统各领风骚，其中比较著名的就有Red Hat、Fedora、Debian、Ubuntu、OpenSUSE及Gentoo等。系统的不断升级，及时反映了IT行业的最新研究成果与开发技术，这使得Linux成为最流行的操作系统之一。在各种Linux系统发行品牌中，Ubuntu Linux系统尤为引人注目，其发展势头与风靡速度之快，堪称典型。

在不同的Linux系统发行品牌中，Ubuntu Linux的系统界面极具吸引力，但与Linux系统本身相比，其桌面环境提供的功能还是相当有限的，只能略尽部分辅助之力。在Linux系统中，桌面环境提供的任何工具实际上都是基于最基本的系统命令实现的，不管桌面环境怎样丰富和发展，都离不开命令行的支持，Linux系统的强大功能完全表现在命令行中，尤其体现在命令的充分发挥与灵活运用方面。这也说明许多专业人员为什么仍然喜欢使用命令行而非桌面环境访问Linux系统。

因此，除了采用一章的篇幅全面介绍GNOME桌面环境，使读者能够快速领略Linux系统的风范，激发学习Linux系统的兴趣之外，本书主要以命令行界面为主，从最基本的命令行与文件系统基础知识及操作入手，由浅入深、逐步阐述Linux系统的基本概念与原理，使读者能够深入理解与掌握Linux系统。此外，为了便于初学者快速入门，后续各章中也辅予以必要的桌面工具的使用说明。

本书以Ubuntu 9.04/9.10版为基准，对Linux系统的Shell编程、用户管理、进程管理、磁盘空间管理、软件管理、文件系统管理、系统启动与关机、作业调度与系统日志、TCP/IP网络管理与应用等内容进行了深入的讨论。

在本书的例子中，需要用户输入的命令均以黑体字形式给出。其中，命令提示符为#号“#”或加前缀“sudo”者表示只有超级用户才能使用的命令，命令提示符为美元符号“\$”者表示普通用户可以使用的命令。此外，为了保持书面的整洁，命令提示符仅采用简单的美元符号“\$”或注释符“#”，省略了其他提示信息。

尽管Linux系统的发展势头如日中天，但其文档做得并不好，随机手册也不能完全令人满意，有些命令说明非常简略，这也是作者编写本书的出发点之一，希望能够为读者提供一定的帮助。

本书所述内容是作者学习Linux系统的一些经验与体会，如能对读者学习Linux系统有所裨益，将是作者莫大的荣幸。由于时间仓促，且限于作者的水平与能力，如有不当之处，恳请广大读者给予批评指正。

在本书的编写过程中，从写作宗旨的确定，到章节内容的安排，都得到了电子工业出版社相关工作人员的热情鼓励与全力帮助。杨敏敏、庞俊华、张广利、邹浪、陈智建、常勇、朱朝辉、王芳、王奇伟、孙伟、赵东江、黄辰、曾伟玲、刘琪、李宗玉、梁志强、袁伟，以及邸静与邢梦可等也给予了大力协助，在此一并表示感谢。

编著者

目 录

第1章 系统概述与安装	1	2.4.3 计算机	40
1.1 Linux的发展过程	1	2.4.4 磁盘分区	40
1.2 充分利用网上资源	2	2.4.5 移动存储设备	41
1.2.1 Ubuntu官方网站	2	2.4.6 搜索文件	42
1.2.2 GNU网站	3	2.5 系统菜单	42
1.2.3 Linux文档项目网站	3	2.5.1 首选项	43
1.2.4 网上求助	3	2.5.2 系统管理	44
1.3 随时查询随机文档	3	2.6 定制GNOME桌面环境	48
1.3.1 查询命令的简单用法	3	2.6.1 定制面板	48
1.3.2 查询系统参考手册	4	2.6.2 定制桌面背景	48
1.3.3 其他命令求助方法	6	2.6.3 定制菜单面板	49
1.4 安装过程	7	第3章 命令行基础知识	51
1.4.1 前期准备	7	3.1 命令行结构	51
1.4.2 安装过程	9	3.2 后台进程	54
1.4.3 后期维护与更新	18	3.3 标准输入、输出与错误输出	55
第2章 GNOME桌面	20	3.4 I/O重定向	56
2.1 GNOME桌面概述	20	3.5 管道	60
2.1.1 GNOME注册界面	20	3.6 元字符与文件名生成	62
2.1.2 GNOME桌面	21	3.7 转义与引用	65
2.2 GNOME桌面浏览	22	3.8 命令历史	68
2.2.1 GNOME菜单面板	22	3.8.1 fc命令	68
2.2.2 GNOME桌面区	25	3.8.2 history命令	70
2.2.3 GNOME窗口面板	27	3.8.3 重复执行先前的命令	71
2.3 应用程序菜单	28	3.8.4 命令行的编辑与执行	73
2.3.1 Internet	28	3.8.5 命令行补充	74
2.3.2 办公	31	3.9 命令别名	76
2.3.3 附件	33	3.10 作业控制	79
2.3.4 图形	35	3.11 会话记录与命令确认	81
2.3.5 影音	35	3.11.1 保存会话记录	81
2.3.6 游戏	38	3.11.2 命令的查询与验证	83
2.3.7 Ubuntu软件中心	38	第4章 文件系统基础知识	85
2.4 位置菜单	38	4.1 文件系统的层次结构	85
2.4.1 主文件夹	39	4.1.1 树形结构	85
2.4.2 桌面、文档等	39		

4.1.2 路径名	86	5.14 从系统中检索文件	127
4.2 文件系统的组织结构	87	5.14.1 简单检索	128
4.3 文件的类型	91	5.14.2 使用逻辑运算符	129
4.3.1 普通文件	91	5.14.3 调用其他命令 处理检索结果	129
4.3.2 目录文件	93	5.14.4 利用管道实现 其他处理功能	130
4.3.3 特殊文件	95	5.15 检索文件内容	130
4.3.4 链接文件	98	5.15.1 利用grep检索文件内容	130
4.3.5 符号链接文件	99	5.15.2 过滤其他命令的输出数据 ..	131
4.3.6 管道文件	101	5.15.3 同时检索多个文件	132
4.4 文件的安全保护机制	101	5.15.4 检索不包含特定 模式的文本行	132
4.4.1 显示文件的访问权限	102	5.15.5 使用正则表达式进行检索 ..	132
4.4.2 修改文件的访问权限	103	5.15.6 检索元字符本身	135
4.4.3 设置文件的访问权限	104	5.15.7 在命令行中使用引号	135
4.4.4 其他访问权限设置	106	5.16 排序	136
第5章 文件与目录操作	108	第6章 vim编辑器	139
5.1 创建文件	108	6.1 启动vim编辑器	139
5.2 显示文件列表	109	6.1.1 创建文件	139
5.2.1 使用ls命令列出文件	109	6.1.2 状态行	140
5.2.2 利用通配符显示文件	111	6.2 vim编辑器的工作模式	140
5.2.3 显示隐藏文件	112	6.2.1 输入模式	141
5.2.4 递归地列出文件	113	6.2.2 命令模式	141
5.3 显示文件内容	114	6.3 保存文件与退出vim	141
5.3.1 使用cat命令显示文件	114	6.4 vim编辑器的基本命令	142
5.3.2 使用more命令分页 显示文件	115	6.4.1 移动光标位置	143
5.3.3 使用less命令分页显示文件 ..	116	6.4.2 输入文本	144
5.3.4 显示文件前几行内容	117	6.4.3 修改与替换文本	144
5.3.5 显示文件最后几行内容	118	6.4.4 撤销先前的修改	145
5.4 复制文件	118	6.4.5 删除文本	145
5.5 移动文件	119	6.4.6 复制、删除与粘贴文本	146
5.6 删除文件	121	6.4.7 重复执行命令	147
5.7 显示当前工作目录	122	6.5 使用ex命令	147
5.8 改换目录	122	6.5.1 显示行号	147
5.9 创建目录	123	6.5.2 多行复制	148
5.10 移动目录	124	6.5.3 移动文本行	148
5.11 复制目录	124	6.5.4 删除文本行	148
5.12 删除目录	125		
5.13 比较文件之间的差别	126		
<VI>			

6.6 检索与替换	149	7.4.4 逻辑运算符	202
6.6.1 字符串检索	149	7.5 命令行的解释执行过程	203
6.6.2 模式检索	150	7.5.1 读取命令行	204
6.6.3 字符串替换	151	7.5.2 命令历史替换	205
6.7 编辑多个文件	151	7.5.3 别名替换	205
6.7.1 编辑多个文件	151	7.5.4 花括号扩展	205
6.7.2 合并文件与文本行	152	7.5.5 波浪号替换	206
6.8 定制vim编辑器的运行环境	152	7.5.6 I/O重定向	207
6.8.1 临时设定vim运行环境	152	7.5.7 变量替换	208
6.8.2 永久定制vim运行环境	155	7.5.8 算术运算结果替换	208
6.9 其他说明	155	7.5.9 命令替换	208
6.9.1 删除或替换特殊字符	155	7.5.10 单词解析	209
6.9.2 在编辑期间运行Linux命令 ..	156	7.5.11 文件名生成	210
6.10 vim编辑器命令总结	157	7.5.12 引用字符处理	210
第7章 Shell基础知识	161	7.5.13 进程替换	211
7.1 Shell与Shell脚本	161	7.5.14 环境处理	212
7.1.1 为什么需要Shell编程	161	7.5.15 执行命令	212
7.1.2 何为Shell脚本	162	7.5.16 跟踪执行过程	213
7.1.3 运行Shell脚本	163	第8章 Shell高级编程	214
7.1.4 退出与出口状态	163	8.1 if条件语句	214
7.1.5 调用指定的Shell解释程序 ..	165	8.1.1 if语句的基本形式	214
7.1.6 位置参数	167	8.1.2 嵌套的if语句	216
7.2 变量与变量替换	169	8.1.3 if语句综合应用实例	218
7.2.1 变量分类	169	8.2 case分支语句	219
7.2.2 变量赋值	170	8.3 for循环语句	222
7.2.3 内部变量	170	8.4 while循环语句	226
7.2.4 变量的引用与替换	173	8.5 until循环语句	227
7.2.5 变量的间接引用	175	8.6 select循环语句	228
7.2.6 特殊的变量替换形式	176	8.7 嵌套的循环	230
7.2.7 变量声明与类型定义	179	8.8 循环控制与辅助编程命令	231
7.3 命令与命令替换	180	8.8.1 break和continue命令	231
7.3.1 Shell内部命令	180	8.8.2 true命令	233
7.3.2 部分命令介绍	184	8.8.3 sleep命令	234
7.3.3 命令替换	195	8.8.4 shift命令	234
7.4 test语句	197	8.8.5 getopt命令	235
7.4.1 文件测试运算符	198	8.8.6 getopts命令	236
7.4.2 字符串测试运算符	200	8.9 循环语句的I/O重定向	239
7.4.3 整数测试运算符	201	8.9.1 while循环的I/O重定向	239

8.9.2	until循环的I/O重定向	240	9.6.5	直接使用root注册	303
8.9.3	for循环的I/O重定向	240	9.6.6	以其他用户身份访问系统 ...	304
8.10	Here文档	241	第10章	进程管理	306
8.11	Shell函数	246	10.1	ps命令概述	306
8.12	逻辑并列结构	252	10.2	查询进程及其状态信息	309
8.12.1	逻辑与命令并列结构	252	10.2.1	查询当前活动的进程	309
8.12.2	逻辑或命令并列结构	253	10.2.2	查询系统中的所有进程	309
8.13	Shell数组	253	10.2.3	显示进程的重要状态信息 ..	310
8.14	信号的捕捉与处理	257	10.2.4	显示进程的详细状态信息 ..	310
8.15	其他Shell课题	261	10.2.5	显示进程间的调用关系	311
8.15.1	子Shell	261	10.2.6	pstree命令	312
8.15.2	Shell脚本的调试	262	10.3	监控进程及系统资源	313
8.15.3	系统性能考虑	267	10.4	终止进程的运行	318
第9章	用户管理	269	10.5	调整分时进程的优先级	320
9.1	增加与删除用户	269	10.5.1	nice命令	321
9.1.1	passwd文件	269	10.5.2	renice命令	322
9.1.2	shadow文件	271	10.5.3	调整进程优先级的作用	323
9.1.3	用户管理实例	272	第11章	proc文件系统	324
9.2	定制用户的工作环境	277	11.1	进程内存映像文件	324
9.2.1	选择命令解释程序	277	11.2	系统配置信息	328
9.2.2	设置用户初始化文件	279	11.3	系统运行状态信息	332
9.2.3	定制Shell工作环境	281	11.4	系统可调参数	337
9.3	增加与删除用户组	287	11.4.1	文件系统可调参数	337
9.4	监控用户	288	11.4.2	系统内核可调参数	338
9.4.1	利用who命令查询用户	288	11.4.3	sysctl命令	342
9.4.2	利用finger命令查询用户	289	第12章	磁盘空间管理	345
9.4.3	利用w命令查询用户活动	290	12.1	查询磁盘空间信息	345
9.4.4	向注册用户发送消息	290	12.1.1	常用磁盘空间管理工具	345
9.5	插件式认证模块	291	12.1.2	使用df命令查询 空间使用情况	345
9.5.1	配置文件、模块类型 与控制标志	291	12.1.3	使用du命令查询 已用存储空间	348
9.5.2	修改PAM配置文件	295	12.1.4	使用find命令找出 超大文件	349
9.6	超级用户与sudo命令	296	12.1.5	使用find命令找出 闲置文件	350
9.6.1	超级用户的访问控制	296			
9.6.2	利用sudo运行特权命令	297			
9.6.3	sudoers配置文件	299			
9.6.4	admin用户组成员 的访问权限	303			

12.1.6 使用find命令处置core文件 ..	351	14.2 创建文件系统	403
12.1.7 使用ls命令检测文件的大小 ..	351	14.2.1 mkfs与mke2fs命令介绍	403
12.2 采用标准工具备份与恢复数据 ..	352	14.2.2 创建Ext2/3/4文件系统	405
12.2.1 利用cpio命令实现		14.3 调整文件系统	406
数据备份与恢复	353	14.4 安装与卸载文件系统	409
12.2.2 利用tar命令实现		14.4.1 安装文件系统概述	409
数据备份与恢复	359	14.4.2 mount命令	410
12.2.3 利用dd命令实现		14.4.3 fstab文件	411
数据的原样复制	368	14.4.4 安装文件系统	412
12.3 采用专用工具备份与恢复数据 ..	370	14.4.5 卸载文件系统	415
12.3.1 利用dump命令备份数据	371	14.5 检测与修复文件系统	417
12.3.2 利用restore命令恢复数据	373	14.5.1 何时需要检测文件系统	418
第13章 软件管理	377	14.5.2 文件系统检测的内容	419
13.1 软件管理概述	377	14.5.3 交互检测与修复文件系统 ..	423
13.1.1 软件维护工具	377	14.5.4 自动检测与修复文件系统 ..	424
13.1.2 软件管理基本概念	377	14.5.5 恢复严重受损的超级块	425
13.2 利用apt-get管理软件包	379	14.5.6 其他文件系统修复方法	426
13.2.1 安装软件包	381	14.5.7 fsck的处理方式	426
13.2.2 软件更新与系统升级	382	14.6 调试文件系统	430
13.2.3 删除软件包	383	14.6.1 概述	430
13.2.4 安装存储介质中的软件包 ..	384	14.6.2 交互调试子命令	430
13.2.5 sources.list配置文件	384	14.6.3 恢复误删的文件	436
13.3 利用aptitude管理软件包	386	14.6.4 恢复误删的文件(续)	438
13.3.1 安装软件包	388	第15章 系统启动与关机	442
13.3.2 更新与升级	389	15.1 磁盘分区与GRUB	442
13.3.3 查询软件包	389	15.1.1 磁盘分区	442
13.3.4 检索软件包	390	15.1.2 GRUB引导程序	443
13.3.5 删除软件包	392	15.1.3 GRUB配置文件	444
13.4 synaptic软件管理工具	392	15.1.4 GRUB实用程序	452
13.4.1 浏览软件包	394	15.1.5 安装或修复GRUB	455
13.4.2 安装软件包	394	15.2 初始引导过程概述	457
13.4.3 删除软件包	395	15.3 系统生成过程	459
13.4.4 软件升级	396	15.3.1 作业配置文件	460
13.5 Ubuntu软件中心	397	15.3.2 rc-sysinit.conf作业	465
13.6 软件包的更新	399	15.3.3 init进程与/etc/init目录	466
第14章 文件系统管理	401	15.3.4 init进程与/etc/rcN.d目录	468
14.1 划分磁盘分区	401	15.3.5 启动应用程序	470
		15.4 login进程	471

15.4.1 login进程与passwd文件	471	17.2 主机名字解析	502
15.4.2 Shell进程与profile文件	471	17.3 网络路由设置	503
15.5 系统关机过程	471	17.4 配置网络服务	504
15.5.1 使用shutdown命令 关闭系统	471	17.5 网络管理与维护	506
15.5.2 使用init命令关闭系统	472	17.5.1 使用ifconfig命令 维护网络接口	506
15.5.3 使用其他命令关机	472	17.5.2 使用netstat命令 监控网络状态	507
第16章 作业调度与系统日志	473	17.5.3 使用ping命令测试 远程主机的连通性	513
16.1 定时运行后台作业	473	17.5.4 使用ping命令检测 网络主机的性能	515
16.1.1 cron守护进程的调度过程 ..	473	17.5.5 使用ftp命令检测网 络主机的传输性能	515
16.1.2 at作业与atd守护进程	474	17.5.6 使用traceroute命令 跟踪路由信息	516
16.1.3 调度错过执行时间的任务 ..	475	第18章 TCP/IP网络应用	518
16.2 调度重复执行的任务	476	18.1 OpenSSH	518
16.2.1 crontab文件及其工作原理 ..	476	18.1.1 安装OpenSSH服务器	518
16.2.2 创建和编辑crontab文件	478	18.1.2 sshd_config配置文件	519
16.2.3 显示crontab文件	479	18.1.3 使用SSH注册到远程系统 ..	522
16.2.4 删除crontab文件	480	18.1.4 执行远程系统命令	523
16.2.5 crontab命令的访问控制	480	18.1.5 使用SCP替代FTP	523
16.2.6 数据库定时备份实例	481	18.1.6 使用SFTP替代FTP	524
16.3 调度一次性执行的作业	482	18.1.7 SSH与SCP的无密码注册 ...	525
16.3.1 提交at作业	483	18.1.8 OpenSSH的安全考虑	527
16.3.2 显示at作业及作业队列	484	18.2 Telnet远程注册	528
16.3.3 删除at作业	484	18.3 FTP文件传输	530
16.3.4 at命令的访问控制	485	18.3.1 设置vsftpd	531
16.3.5 系统定时关机实例	485	18.3.2 vsftpd.conf配置文件	531
16.4 系统日志	487	18.3.3 ftp命令	535
16.4.1 系统日志文件	487	18.3.4 FTP应用	537
16.4.2 应用程序日志文件	488	18.3.5 FTP自动注册	538
16.4.3 无法直接查阅的日志	489	18.3.6 FTP安全考虑	539
16.4.4 系统日志守护进程	490		
第17章 TCP/IP网络管理	493		
17.1 网络接口设置	493		
17.1.1 以太网网络设置	493		
17.1.2 ADSL网络连接	499		

第1章 系统概述与安装

作为本书的开始，本章首先将简单地介绍Linux的发展过程，概述Linux的各种网络资源与求助方法，最后详细介绍Linux系统的安装过程。

1.1 Linux的发展过程

提到Linux的缘起与发展过程，不能不涉及UNIX。UNIX系统早期之所以能够取得巨大的成功并迅速得到普及，主要在于其三个重要特点：简洁性、开放性与可移植性。向大学和研究机构公开源代码，激发了软件开发人员研究和移植UNIX系统的兴趣，导致UNIX成为操作系统的新宠；许多大学均以UNIX作为操作系统课程的研究对象，从而出现了《UNIX操作系统设计》等著名的UNIX教材。这使得UNIX成为大学操作系统课程的代名词，同时也培养了许多潜在的UNIX系统准用户。

而后期的商业化运作方式，使得UNIX系统及其源代码成为专属产品，限制了软件人员对UNIX系统的研究、开发和使用。另外，为了考虑特定的机器结构，商业化的UNIX也开始变得越来越复杂，基本上失去了可移植性的特点。而这一切因素导致了开源软件运动的兴起，其中的一个结果就是催生了Linux。

1984年，Richard Stallman（UNIX系统emacs编辑器的开发者）发起了一场自由软件共享活动，创建了一个非营利性的自由软件基金会（Free Software Foundation），支持开发共享自由软件。其中的GNU项目旨在开发一个完全免费的、类似于UNIX的GNU操作系统，但不使用UNIX系统的任何源代码。Stallman希望通过社区参与的方式，促进GNU操作系统的发展，使用户能够自由交流、学习，从而改进或不断增强这一系统。由于开发一个完整的操作系统（包括内核与实用程序）是一项十分艰巨的任务，GNU决定采用模块化的设计方法，以便任何人都能够参与进来共同开发各个操作系统模块，且能够非常容易地集成现有的自由软件。到了1990年，针对UNIX系统的所有实用程序、工具与核心库函数，GNU几乎都有了自己的相应软件，其中包括emacs文本编辑器以及C编译器gcc等，但缺乏一个内核。

与此同时，1991年尚在芬兰赫尔辛基大学读书的Linus Torvalds决定在个人计算机上创建一个新的、类似于UNIX操作系统的内核。Torvalds一直使用由Andrew Tannenbaum设计与实现的Minix操作系统，因而熟悉UNIX系统的功能特性。Torvalds决定开发一个可在个人计算机上运行的UNIX系统，并于1991年9月推出了Linux 0.01版。由于开发一个高质量的操作系统非一人之力所能及，于是，Torvalds利用Internet对外公开其源代码，任何人都可以免费下载和使用。Torvalds邀请其他人下载其新内核的副本，帮助改善和增加新的功能特性。此举立即引起世界各地软件开发人员的极大兴趣，许多人决定接受Torvalds的提议，开始参与Linux的开发与传播。作为一个团队，他们分工合作，改进Linux，从而扩展了Linux内核，开发出许多系统程序和工具软件，把BSD与System V版UNIX的许多功能加到新的Linux系统中，从而构成了一个完整的操作系统。

组合了GNU软件的Linux（有时也称做GNU/Linux）包含类似于UNIX的实用程序、工具、核心库、编译器、文本编辑器、桌面环境以及其他组成部分，构成了一个完整的UNIX系统环境。

从开始之日起，Linux的所有开发工作一直都是在Torvalds的指导下，利用Internet相互交流，共同合作完成的。Linux系统是世界各地许多软件开发人员共同努力的结果，也是借助于Internet协同开发的产品。Linux是一种免费的操作系统，所有的软件，包括源代码、文档和技术支持（通过Internet）都是免费的。任何人均可自由获取源代码、研究、修改和重新发行，而这一切都是免费的。

目前，存在许多不同版本的Linux产品，其中比较著名的有Red Hat、Fedora、Debian、Ubuntu、OpenSUSE以及Gentoo等。尽管这些系统在安装过程和外部表现等方面有所不同，但其内部采用的Linux内核、标准的实用程序等基本上是一致的，因而具有许多共性。

在20世纪90年代，南非的Mark Shuttleworth参与了Debian Linux系统的开发。2004年3月份，他开始转向自己的自由软件世界，成立了Canonical公司，决定开发一个基于Debian的、用户友好的Linux系统，并以此公司作为技术支撑，提供服务。Ubuntu Linux系统迅速崛起，致使Shuttleworth又于2005年投资1000万美元成立了Ubuntu基金会，专门致力于Ubuntu Linux系统的开发与推广，确保Ubuntu Linux系统未来的健康发展。

Ubuntu Linux系统是众多Linux发行品牌之一。在Linux世界中，Ubuntu只是一个后来者，是一个非常年轻的Linux发行品牌，用了短短几年的时间就发展成为一个流行的、成熟的及桌面环境丰富的Linux系统，受到了从Linux初学者到资深专家的大批Linux用户追捧。

Ubuntu是一个古老的非洲词汇，表示人类之间的关怀、共享、和谐。作为一种理念，它倡导个人、文化以及民族之间的融合、博爱与相互合作。

1.2 充分利用网上资源

前面曾经说过，Linux系统本身是一种“Internet产品”，网上积累了大量的Linux资源与信息，其中包括Ubuntu Linux自己的官方网站、GNU自由软件项目、Linux文档项目、Linux专题讨论组（Newsgroups）、电子邮件通信录（Mailing Lists），以及各种各样的Linux社区（包括Ubuntu社区）与论坛。

用户可以充分利用Internet，查阅相关的文档，寻求问题或故障解决方案，也可以上网介绍自己的学习心得，与他人分享自己的学习经验等。

1.2.1 Ubuntu官方网站

Ubuntu Linux系统官方网站的地址为<http://www.ubuntu.com>（相应的中文网站地址为<http://www.ubuntu.org.cn>），其中包含怎样获取Ubuntu、怎样获得支持、Ubuntu简介等栏目，以及Ubuntu桌面版与服务器版的下载链接。可以以此作为出发点，获取命令参考手册、文档与求助信息，也可以直接访问下列网址，从而获取相关的资讯。

- Ubuntu求助中心（<https://help.ubuntu.com>）——其中提供各种版本的Ubuntu Linux系统文档，如9.10版的安装文档等。如果想要了解怎样获取帮助，Ubuntu都提供什么可用的资源，也可以访问<https://wiki.ubuntu.com/HowToGetHelp>网站，以达到解惑释疑之目的。

- Ubuntu技术支持 (<http://www.ubuntu.com/support>) ——可以从此窗口进入Ubuntu社区和论坛, 寻求问题答案, 获取技术文档, 了解Ubuntu提供的各种服务等。
- Ubuntu社区 (<http://www.ubuntu.com/community>) ——可以借此途径参与软件开发或界面设计等。其中的文档区 (<https://help.ubuntu.com/community/UserDocumentation>) 提供Ubuntu社区贡献的大量技术文档, 包括系统安装、系统管理、服务器管理 (如MySQL数据库、DNS、Apache、OpenSSH、NFS与Samba等) 以及故障修复等。
- Ubuntu中文维客网站 (<http://wiki.ubuntu.org.cn>) ——是一个收集、整理、翻译以及编写Ubuntu Linux系统中文文档, 相互讨论与交流的场合, 其中分门别类, 提供了大量的中文文档, 也欢迎志愿者参与文档的翻译与审校工作。
- 用户通信 (<https://lists.ubuntu.com>) ——利用电子邮件, 可以向Ubuntu同行或爱好者请教问题, 获取建议, 解答问题等。

1.2.2 GNU网站

<http://www.gnu.org>是GNU项目的大本营, 其中的<http://www.gnu.org/manual>提供了许多命令参考手册与文档, 如bash、finger、gawk、grep、info、sed以及tar等。

1.2.3 Linux文档项目网站

Linux系统文档的一个重要来源是“Linux文档项目”网站。若想查阅Linux文档, 可以利用浏览器, 访问“Linux文档项目”网站<http://www.tldp.org>。其中的HOWTO链接提供许多课题的帮助信息, 如KernelAnalysis-HOWTO、LinuxLoadable Kernel Module HOWTO以及The Linux Bootdisk HOWTO等。Guide链接包含许多PDF文档, 如Linux System Administrator Guide、Advanced Bash-Scripting Guide以及Linux Kernel 2.4 Internals等。

1.2.4 网上求助

在日常的学习与上机过程中, 如果遇到问题, 可以利用Google等搜索引擎, 从Internet中积累的庞大Linux资源中寻求答案。例如, 为了查询怎样禁用IPv6, 以减少不必要的系统开销, 提高系统与网络性能, 可在Google中搜索“disable ipv6 ubuntu 9.10”等。

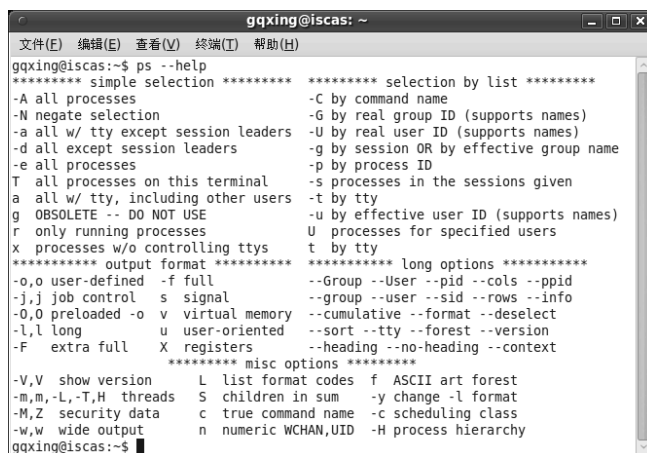
1.3 随时查询随机文档

1.3.1 查询命令的简单用法

大多数GNU命令都提供“-help”选项, 用于显示相应命令用法的简单说明, 对于熟悉和快速访问Linux系统具有极大的帮助, 如图1-1所示。

如果帮助信息太长, 超过一个终端窗口能够显示的内容, 可以利用管道机制, 以及less或more命令, 逐页显示命令的帮助信息 (详见第3章“命令行基础知识”与第5章“文件与目录操作”)。

```
$ ls --help | more
```



```
gqxing@iscas: ~  
gqxing@iscas:~$ ps --help  
***** simple selection *****  
-A all processes  
-N negate selection  
-a all w/ tty except session leaders  
-d all except session leaders  
-e all processes  
T all processes on this terminal  
a all w/ tty, including other users  
g OBSOLETE -- DO NOT USE  
r only running processes  
x processes w/o controlling ttys  
***** output format *****  
-o,o user-defined -f full  
-j,j job control s signal  
-O,O preloaded -o v virtual memory  
-l,l long u user-oriented  
-F extra full X registers  
***** selection by list *****  
-C by command name  
-G by real group ID (supports names)  
-U by real user ID (supports names)  
-g by session OR by effective group name  
-p by process ID  
-s processes in the sessions given  
-t by tty  
-u by effective user ID (supports names)  
U processes for specified users  
t by tty  
***** long options *****  
--Group --User --pid --cols --ppid  
--group --user --sid --rows --info  
--cumulative --format --deselect  
--sort --tty --forest --version  
--heading --no-heading --context  
***** misc options *****  
-V,V show version L list format codes f ASCII art forest  
-m,m,-L,-T,H threads S children in sum -y change -l format  
-M,Z security data c true command name -c scheduling class  
-w,w wide output n numeric WCHAN,UID -H process hierarchy  
gqxing@iscas:~$
```

图1-1 使用命令的“--help”选项

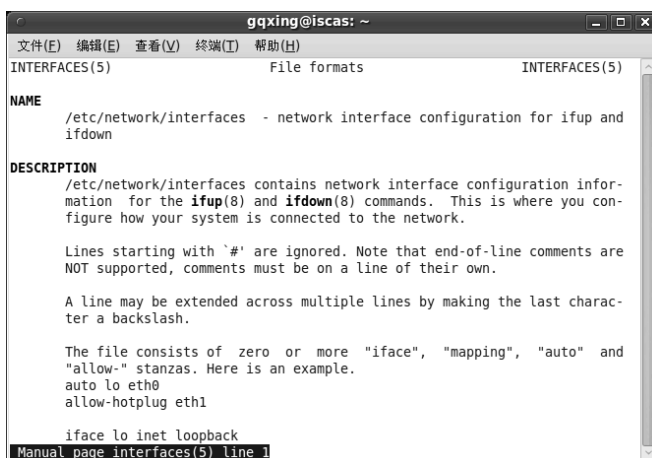
1.3.2 查询系统参考手册

Linux系统提供大量的系统命令和用户命令，而许多命令又有众多的选项，一个人不可能记住这么多命令，即使具有多年Linux系统经验的资深人员也是如此。对于常用的命令，也不可能记住每一个选项。为此，Linux系统随机提供了一套完整的命令参考手册，供用户随时查阅命令的说明与用法。

同UNIX系统一样，Linux系统的参考手册也是通过man命令实现的。为了查询Linux系统提供的任何命令，了解命令的功能、用法、可用的选项以及参数，可以使用下列命令（其中的command可以是任何Linux命令，包括man命令本身）：

```
$ man command
```

Linux系统中的man命令借助于less命令，逐页显示指定命令的参考手册，并在终端窗口的左下角输出一个冒号“:”提示符。此时，可以使用空格键逐页显示余下的解释内容，或使用“q”子命令退出man命令。因此，当使用man命令查询系统参考手册时，可以参照第5章“文件与目录操作”中介绍的less命令，按照less命令的用法逐页显示，或随机翻阅，如图1-2所示。



```
gqxing@iscas: ~  
INTERFACES(5) File formats INTERFACES(5)  
NAME  
/etc/network/interfaces - network interface configuration for ifup and ifdown  
DESCRIPTION  
/etc/network/interfaces contains network interface configuration information for the ifup(8) and ifdown(8) commands. This is where you configure how your system is connected to the network.  
Lines starting with '#' are ignored. Note that end-of-line comments are NOT supported, comments must be on a line of their own.  
A line may be extended across multiple lines by making the last character a backslash.  
The file consists of zero or more "iface", "mapping", "auto" and "allow-" stanzas. Here is an example.  
auto lo eth0  
allow-hotplug eth1  
iface lo inet loopback  
Manual page interfaces(5) line 1
```

图1-2 使用man命令查阅参考手册

对于任何用户而言, 利用man命令随时查询命令的帮助信息是非常有用的。对于常用的命令, 即使忘记了不常用的命令选项或用法, 当只知其名, 不知其用法, 或只知其意, 不知其名 (参见后面即将介绍的“man -k”命令) 时, 均可借助于man命令获取帮助信息。

由于Linux系统的参考手册非常庞大, 为了便于用户快速准确地查阅随机文档, 按照UNIX系统的传统, 以及文件系统层次组织标准 (Filesystem Hierarchy Standard, FHS), Linux系统把各种参考手册的内容至少分为10节, 其中的8节分别对应下列8种文档类型。

- (1) 用户级的命令、Shell脚本和应用程序;
- (2) 系统调用 (系统内核提供的函数);
- (3) 库函数;
- (4) 特殊文件 (通常是位于/dev目录中的设备文件);
- (5) 系统文件格式定义;
- (6) 游戏;
- (7) 系统数据定义文件 (如信号和时间等)、网络协议 (如IP、TCP与UDP等) 以及字符集 (如UTF-8、unicode与iso-8859等);
- (8) 系统管理与维护等特权命令 (通常是仅适用于超级用户root的命令等)。

chmod、chown、kill、mount、nice与uname等既是Linux系统中的命令, 也是同名的系统调用。passwd、locale与crontab等既是命令名, 又是系统文件名 (或一类文件的总称)。如果不加区别, man命令通常只会给出命令的说明, 而不会输出系统调用或系统文件格式等说明。因此, 为了查询passwd系统文件, 可以使用下列命令 (在查询的对象之前, 指定对象所属的文档类别):

```
$ man 5 passwd
```

当用户需要完成某种任务, 而不知道究竟应使用哪一个命令及其确切的名字时, 可以在man命令中使用“-k”选项, 按照关键字进行检索。“man -k”命令将会利用给定的关键字检索所有联机手册中每个命令的简单描述部分, 显示匹配的命令及其简单说明。

下面的例子说明了怎样使用“man -k”命令查询关键字locale, 其输出结果包括每个与之相关的命令、系统调用、库函数、文件、文档类型及简单描述。

```
$ man -k locale
locale (1)           - Get locale-specific information.
locale (5)           - Describes a locale definition file
locale (7)           - Description of multi-language support
locale-gen (8)       - compile a list of locale definition files
locale.alias (5)     - Locale name alias data base
Locale::gettext (3pm) - message handling functions
localedef (1)        - compile locale definition files
luit (1)             - Locale and ISO 2022 support for Unicode terminals
lxtterm (1)          - locale-sensitive wrapper for xterm
update-locale (8)    - Modify global locale settings
validlocale (8)      - Test if a given locale is available
$
```

所有的man随机文档均位于/usr/share/man目录中。由于man随机文档是采用特殊的nroff格式实现的, 无法直接打开阅读。为了获得文本格式的文档, 以便复制或下载到笔记本电脑供随时

查阅，可以使用col命令剔除其中的格式控制字符。例如，为了生成一个文本格式的find命令说明文件，可以使用下列命令：

```
$ man find | col -b > find
$
```

1.3.3 其他命令求助方法

作为一种补充手段，还可以借助于info命令，获取命令的说明信息。info是一个由GNU项目开发，且随Linux系统一同发行的实用程序，也是一个基于菜单选择的超文本帮助工具。info命令本身包含一个自学功能（也可以使用“info info”命令，或访问<http://www.gnu.org/software/texinfo/manual/info>网站，获取使用说明），其中提供Shell、系统实用程序以及应用程序的说明文档。当使用man命令无法获取某个命令（如cpio命令）的使用说明时，可以借助于info命令。例如：

```
$ info cpio
```

图1-3给出的是直接执行info命令后的输出结果。此时，可以通过下列部分按键获取各种命令的帮助信息。

- h或?：查询info命令的用法，了解其他按键的作用。
- 空格：向下滚动屏幕，以便逐屏查阅更多信息。
- Del：向前（起始查询位置）滚动屏幕。
- d：返回info命令查询的起始位置。
- m：输入m和任何一个命令，能够快速定位一个欲查询的命令。
- q：退出info命令。



图1-3 执行info命令后的输出结果

在info列出的命令查询菜单中，每个命令占用一行，除星号“*”之外，第一列是各种Linux命令的名字，括号内的内容是命令所属的软件包，最后一列是命令的简要说明。

在info命令查询菜单中，每个菜单项实际上是一个链接，指向相应命令的说明文档。若想查阅某个命令，可以把光标移至相应的命令项，然后按Enter键。也可以直接输入“m cmd”，接着按下Enter键，其中的cmd是命令的名字。例如，为了查询cpio命令的说明，可以输入

“mcpio”，然后按Enter键。在输入字符m之后，光标将会立即移至终端窗口左下角最后一行的起始位置，同时显示“Menu item（菜单项）：”提示信息，输入cpio，接着按下Enter键之后，info将会显示选定的命令说明信息。

1.4 安装过程

Ubuntu Linux系统支持不同CPU类型的计算机，其中包括Intel x86系列处理器及其兼容机、PowerPC处理器，以及64位的Intel/AMD处理器等，几乎可以安装到所有的台式机、笔记本电脑以及服务器中，其安装方式与安装过程极其灵活，即使先前没有Linux方面知识的用户，也能够毫无困难地安装一个Linux学习环境。

Ubuntu Linux系统既可以单独安装，也可以与Microsoft Windows系统安装在同一台计算机上，把Ubuntu Linux安装到Windows系统未用的磁盘分区中。注意，在安装Ubuntu Linux与Windows双系统时，应首先安装Windows，然后再安装Ubuntu Linux系统，否则无法正常启动Ubuntu Linux系统。

此外，在安装Windows系统时，必须为Ubuntu Linux系统预留出磁盘空间。如果Windows系统已经分为多个逻辑盘，如C和D两个逻辑盘，需要事先删除一个逻辑盘（如D盘），用于安装Ubuntu Linux系统。否则，不管Windows系统划分的磁盘分区是否已经使用，都无法安装Ubuntu Linux系统，除非清除原先安装的Windows系统。因此，如果想把Linux系统安装到D盘，可以选择“管理工具→计算机管理→磁盘管理”，在磁盘窗口中右击D盘，然后从上下文菜单中选择“删除逻辑驱动器”。

1.4.1 前期准备

1. 硬件要求

在安装Ubuntu Linux系统时，不同的系统与版本对硬件的要求不尽相同。表1-1以Intel x86系列机和桌面版的Ubuntu 9.10 Linux系统为例，给出了一个基本硬件要求（其中包括CPU、内存及磁盘空间等需求），供选择计算机系统时参考。

表1-1 硬件系统要求

硬件系统要求	简单说明
CPU	至少选用1.0 GHz的Pentium 4 CPU，建议采用更快的Intel x86系列CPU
内存	至少配备256 MB（字符界面）或384 MB内存（图形界面）；建议配备512 MB或更多的内存
磁盘或磁盘分区	标准桌面系统本身需要5 GB的存储空间，完整桌面系统本身需要4 GB的存储空间。此外还要考虑预留两倍内存容量的磁盘空间作为交换分区，以及用户数据的空间需求。因此，完整的安装至少需要8 GB的磁盘空间
VGA显卡/显示器分辨率	分辨率1024×768像素
引导设备	CD/DVD驱动器，或者采用USB移动盘、内置硬盘或其他安装方式

2. 磁盘分区

安装Ubuntu Linux系统时，至少需要两个磁盘分区，分别用于创建“/”文件系统与交换分区。对于初学者个人使用的Ubuntu Linux系统而言，最简单或最佳的选择是重新划分Windows系统中的D盘（或其他盘等），使之分为两个分区，较小的分区用做交换分区，较大的分区用做“/”文件系统。当然，也可以在安装Ubuntu Linux系统时重新划分D盘。

如果磁盘分区容量较大，如大于6 GB，设置磁盘分区类型时应选择Ext3或Ext4文件系统。由于Ext2文件系统需要周期地执行完整性检测，故磁盘分区较大时容易引起系统性能下降。

Linux系统采用交换分区提供虚拟内存，因此，交换分区的设置非常重要。在确定交换分区的大小时，通常应以系统配置的内存为参照。如果系统内存小于1 GB，可以把交换分区的容量设为内存容量的两倍。如果内存大于等于1 GB，交换分区的大小可以等于物理内存的容量。但在一个32位的计算机系统中，交换分区的大小不能超过2 GB。如果确实需要使用更多的交换分区，可以设置多个交换分区，如果可能，最好把每个交换分区分布到不同的磁盘中。这种解决方案既克服了交换分区的容量限制，又能够借以实现负载均衡，从而提高系统的性能。

如果是一个多用户系统，且系统配有大量的磁盘存储空间，最好划分多个磁盘分区，在每个磁盘分区创建一个单独的文件系统，如/usr、/var和/home等文件系统。如果磁盘的容量太大，BIOS无法直接访问1023之外的柱面时，还可以增加单独的/boot文件系统分区，但不能把/bin、/dev、/etc、/lib、/root和/sbin等目录作为单独的文件系统分区，这些目录均应位于“/”文件系统分区中。

每个文件系统分区都有一个安装点，表示相应文件系统在整个Linux文件系统目录层次结构中的安装位置。除了单独的文件系统分区，分别用于存储各自的文件或数据之外，其他所有文件或数据均存储在“/”文件系统分区中。

当需要且决定划分多个磁盘分区，以便创建单独的/usr、/var和/home等文件系统时，可以参考表1-2给出的磁盘分区要求与建议。

表1-2 磁盘分区要求与建议

文件系统分区	最小容量要求	建议的容量分配	文件系统类型
/	2 GB	5 GB	Ext3或Ext4
/usr	1.5 GB	4 GB ~ 6 GB	Ext3或Ext4
/var	2 GB ~ 3 GB	或更多（取决于系统是否用做数据库、Apache或电子邮件等服务器）	Ext3或Ext4
/home	2 GB	或更多（取决于用户数量以及用户数据的空间需求）	Ext3或Ext4

3. 安装方式

Ubuntu Linux系统的安装方式极其灵活，可以采用不同的方法引导与安装系统：如下载各种ISO映像文件，采用刻录的CD/DVD安装介质，安装一个基本的Ubuntu Linux系统；利用GRUB引导程序与内置硬盘（或USB移动盘）中的ISO映像文件，直接安装Ubuntu Linux系统；

也可以在Windows系统中安装一个虚拟的Linux系统等。安装过程也比较简单，整个安装过程只需7个步骤即可完成。

1.4.2 安装过程

在上述准备工作完成之后，即可开始安装Ubuntu Linux系统。下面以Ubuntu 9.10的DVD映像文件，384MB内存和14GB Linux系统分区的IBM A22e笔记本电脑为例（C盘装有Windows XP系统），详述Ubuntu Linux系统的安装过程。

1. 下载Ubuntu 9.10映像文件

在开始安装Ubuntu Linux系统之前，首先需要选择一个快速的网站，按照计算机的CPU类型，选择下载一个Ubuntu 9.10 CD/DVD映像文件，如ubuntu-9.10-desktop-i386.iso或ubuntu-9.10-dvd-i386.iso，提取casper目录中的initrd.lz和vmlinuz两个文件，置于C盘根目录，然后把ubuntu-9.10-dvd-i386.iso文件也置于C盘根目录（其他盘的根目录也可）。



除了正常安装之外，CD/DVD映像文件也可用于系统维护，以便在系统无法正常启动时，或忘记超级用户密码等情况下恢复Ubuntu Linux系统。

2. 下载GRUB4DOS引导程序

从GRUB4DOS官方网站<http://download.gna.org/grub4dos>下载一个最新的GRUB4DOS软件，如grub4dos-0.4.4-2009-06-20.zip，抽取其中的GRUB引导程序grldr，置于C盘根目录。

3. 修改boot.ini文件

利用任何文本编辑器，如记事本，修改Windows系统C盘根目录中的boot.ini文件，在“[operating systems]”一节后面附加一行“c:\grldr="Start GRUB for DOS"”引导选项。修改后的boot.ini文件内容如下：

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin /fastdetect
c:\grldr="Start GRUB for DOS"
```

4. 创建menu.lst文件

利用任意文本编辑器，在Windows系统的C盘根目录中创建一个menu.lst文件，其中包含下列内容：

```
timeout 30

title Boot Ubuntu 9.10
root (hd0,0)
kernel (hd0,0)/vmlinuz boot=casper iso-scan/filename=/ubuntu-9.10-dvd-i386.iso ro quiet splash locale=zh_CN.UTF-8
initrd (hd0,0)/initrd.lz
```


5. 重新启动系统

在完成上述准备工作之后，即可重新启动系统，当出现如图1-4所示的引导界面时，使用箭头键选择“Start GRUB for DOS”项。

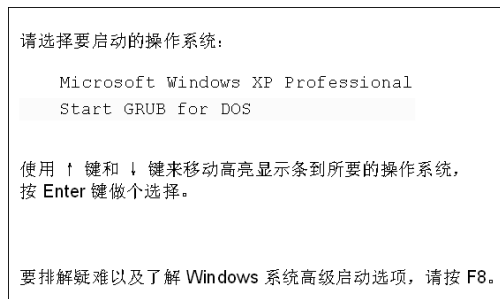


图1-4 引导界面

6. 重新启动系统

当出现如图1-5所示的引导界面时，按下Enter键。

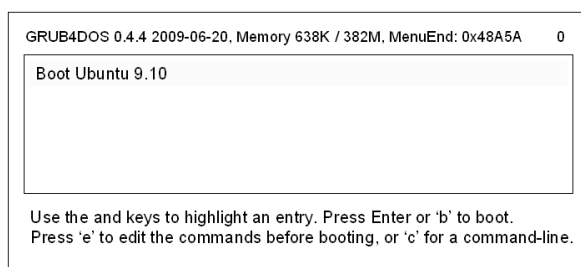


图1-5 Ubuntu引导界面

7. 引导Ubuntu Linux系统

此时开始引导Ubuntu Linux系统，这一过程相当于利用CD/DVD安装介质直接引导系统，最终将会进入一个基于内存的Ubuntu Linux系统，如图1-6所示。

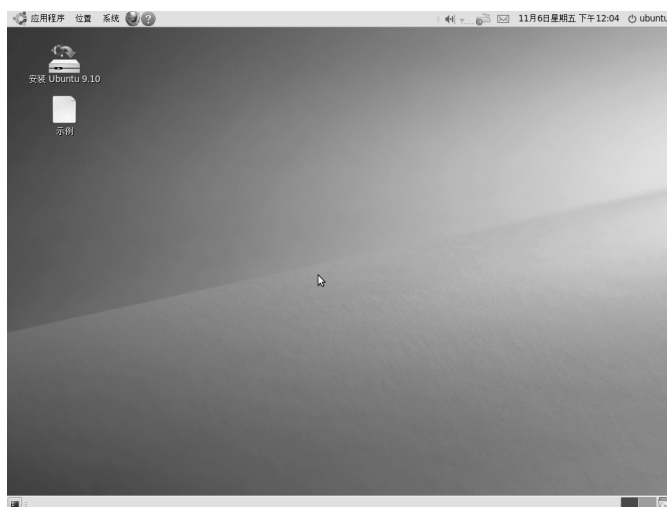


图1-6 基于内存的Ubuntu Linux系统界面

8. 重新启动系统

在GNOME桌面中选择“应用程序→附件→终端”菜单，在新弹出的终端窗口中输入下列命令，以便在必要时卸载系统盘，否则无法修改磁盘分区表，创建Linux系统分区：

```
$ sudo umount -l /isodevice
$
```

然后双击GNOME桌面上的“安装Ubuntu 9.10”图标，开始安装Ubuntu Linux系统。

9. 选择语言环境

当“欢迎”界面出现时，需要选择安装过程使用的语言，选中的语言也将成为最终系统默认的语言环境。如果选用“中文（简体）”，可以单击“前进”按钮继续，如图1-7所示。



图1-7 “欢迎”界面

10. 时区选择

在图1-8所示的“您在什么地方”界面中，可以设置系统的时区。Ubuntu安装程序提供两种时区选择方法：世界地图与下拉菜单。

- 利用世界地图选择时区。根据自己所在的区域，上下左右移动鼠标，选择一个距离自己最近的城市黄点，选中的黄点将会闪烁，同时在下拉框中显示选定的城市，以城市的地理位置表示用户选中的时区。
- 利用“区域”下拉菜单选择城市。选择一个距离自己最近的城市，如“Shanghai”（表示北京时间CST，即GMT+8:00）。

选定城市后也就选定了时区，然后单击“前进”按钮继续。注意，安装过程只能设置时区，不能设置系统的日期与时间。如果系统显示的日期与时间不正确，可在安装结束，注册到系统之后再作调整。

11. 键盘配置

在“键盘布局”界面左边的窗口中，默认的选择为美式键盘（“USA”）。必要时也可以单击“选择您的”单选钮，从Ubuntu Linux系统支持的不同键盘类型中选择其他键盘，如“China”，然后单击“前进”按钮，如图1-9所示。



图1-8 “您在什么地方” 界面

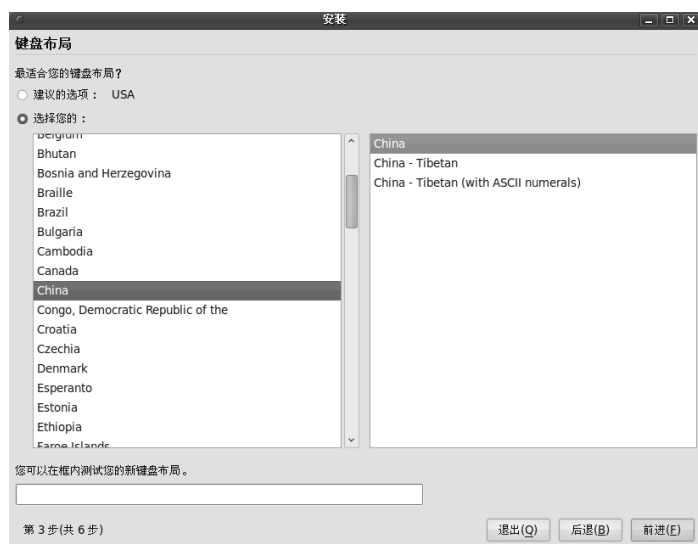


图1-9 “键盘布局” 界面

**注意**

在选定键盘类型之后，可以利用按键测试选定的键盘是否正确。

1.2. 磁盘分区

Ubuntu Linux系统可以安装在任何空闲的磁盘或磁盘分区中。在“准备硬盘空间”界面中，安装程序会给出安装前的磁盘使用与分配情况。如果想在整个磁盘上单独安装Ubuntu Linux系统，安装时可以选择“清空并使用整个硬盘”，由安装程序定义磁盘分区布局。对于资深用户而言，可以选择“手动指定分区”的安装方式，自己选择预留的磁盘分区，或自己划分适当数量的磁盘分区，用于创建单独的/home、/usr、/var或/boot等文件系统，从而设定文件系统布局，如图1-10所示。



图1-10 “准备硬盘空间”界面

进入“手动指定分区”安装方式之后，如果事先只是预留了一个磁盘分区，而未做进一步的细分，需要在此处划分磁盘分区。如果已经在Windows系统中划分了“/”文件系统分区和交换分区，只需在此处指定哪一个分区用做“/”文件系统，哪一个分区用做交换分区，如图1-11所示。

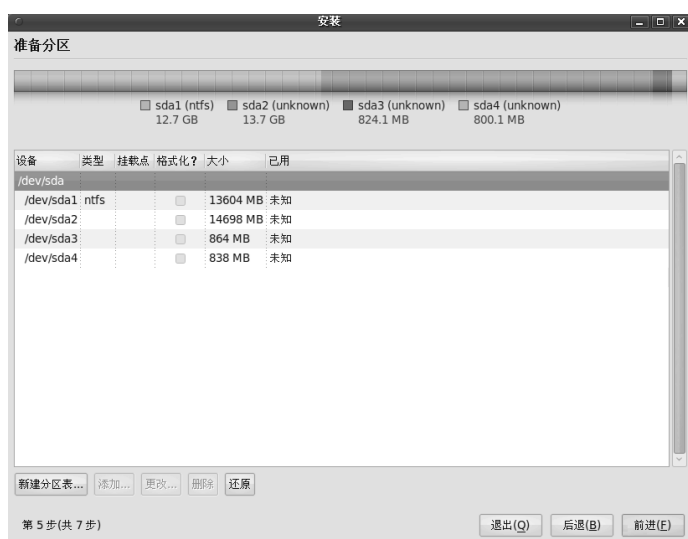


图1-11 “准备分区”界面（1）

在图1-12中，磁盘分区/dev/sda1标记为ntfs，表示这是一个NTFS文件系统分区，也即Windows系统的C盘，故应保持不变。磁盘分区/dev/sda2和/dev/sda3分别用“/”文件系统和交换分区。

使用鼠标选中“dev/sda2”，然后单击“更改”按钮，安装程序将会弹出一个“编辑分区”界面。在“用于”栏目中，选择磁盘分区的文件系统类型，如最新的“Ext4日志文件系统”或“Ext3日志文件系统”等。如果选择的磁盘分区用做文件系统，尚需在“挂载点”栏目中输入文件系统的安装点，如“/”。然后单击“确定”按钮，即可创建一个“/”文件系统分区，如图1-12所示。



图1-12 “准备分区”界面（2）

使用鼠标选中“/dev/sda3”，然后单击“更改”按钮，同样，安装程序又会弹出一个“编辑分区”界面。在“用于”栏目中，选择“交换空间”项，单击“确定”按钮后，即可创建一个交换分区，如图1-13所示。



图1-13 “准备分区”界面（3）

至此，磁盘分区的分配任务已全部完成，可以单击“前进”按钮继续。分区后的最终结果如图1-14所示。

13. 创建用户账号

Ubuntu建议用户总是使用自己的账号而非超级用户root注册Linux系统，以确保系统的安全。在随后出现的“您是谁”界面中，可以为自己创建一个普通用户账号。输入用户名的全称、有效的注册用户名，输入并确认自己的密码即可。此外，还需要设置系统的主机名。如果希望在系统启动之后能够自动注册，直接进入GNOME桌面环境，可以选择“自动登录”选项。最后

单击“前进”按钮，参见图1-15。

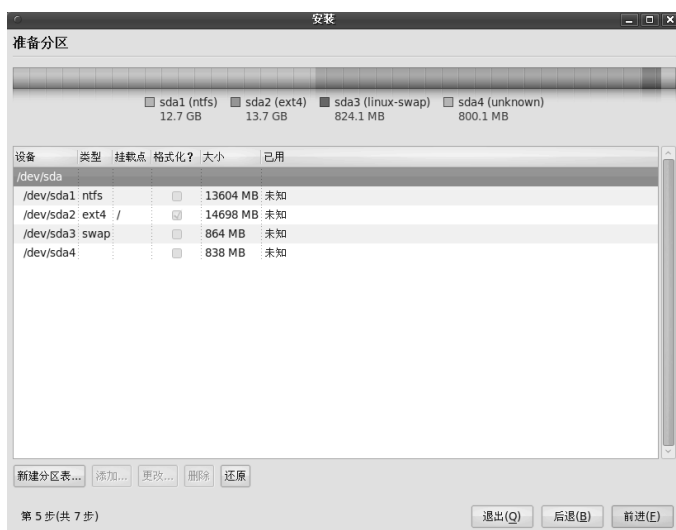


图1-14 “准备分区”界面（4）



图1-15 “您是谁”界面



除了同名的用户组，此时创建的第一个用户也会自动加入到admin用户组中，因而拥有admin用户组成员的访问权限，能够执行一定的系统管理与维护任务。因此，当需要利用超级用户的访问权限执行系统管理任务时，可以在命令行界面中使用sudo命令，即在实际运行的命令前面增加一个sudo前缀，在提示输入密码时，输入此时设置的密码（为了便于叙述，此后我们把安装Ubuntu Linux系统时创建的第一个用户的密码简称做sudo密码）。

14. 恢复数据文件

如果计算机已经安装了Windows系统，或先前已经安装过Ubuntu Linux系统，安装程序将会显示一个“Migrate documents and settings”界面，提供一个恢复用户主目录中的文档和设

置的机会。如果想要导入或恢复其中的数据，可以勾选相应的复选框。否则，直接单击“前进”按钮，进入下一步，如图1-16所示。



图1-16 “迁移文档和设置”界面

15. 准备安装

在“准备开始安装”界面中，安装程序汇总了先前做的一系列选择与设置，如果没有问题，可以单击“安装”按钮正式开始安装Ubuntu Linux系统。如果设置有误，可以单击“后退”按钮，返回先前的界面重新设置，如图1-17所示。



图1-17 “准备开始安装”界面

实际上，在此之前的每一个设置步骤中，如果设置有误，均可单击“后退”按钮，返回先前的界面重新设置。

16. 安装软件包

至此，Ubuntu安装程序开始安装软件，同时在屏幕上显示整个安装的进度。安装程序首先

会格式化磁盘分区（如果先前勾选了“格式化”复选框），创建文件系统，计算文件占用的磁盘空间，复制文件，安装软件包，设置系统，配置apt，检测镜像网站，设定软件源，设置时区，与网络时间服务器同步时间，设置键盘，导入文档和设置（如果先前勾选了导入复选框），检测硬件，加载USB模块，设置硬件，以及删除临时文件等，如图1-18所示。

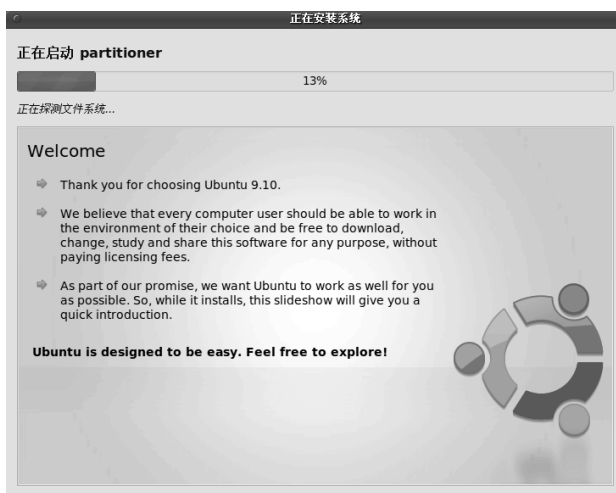


图1-18 安装软件包界面

取决于计算机的配置和网络的速度，这一安装过程将会持续较长的时间，但中间无需用户干预。安装完成之后，安装程序将会执行安装后的软件配置，把GRUB引导程序复制到磁盘中，最后显示如图1-19所示的对话框，表示整个安装过程已经完成。

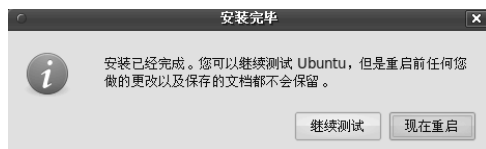


图1-19 “安装完毕”对话框

单击“现在重启”按钮，重新启动系统。在重启之前，系统将会提示用户取出CD/DVD安装介质，然后按Enter键。此时可以忽略此信息，直接按下Enter键。

在成功地重新启动之后，系统将会显示如图1-20所示的注册界面。此时，可以使用先前创建的用户名与密码注册，访问Ubuntu Linux系统了。



不要使用超级用户的账号root注册。在Linux系统中，超级用户root拥有无限的权力，如果使用不当，可能会损害或危及系统的安全，故Ubuntu Linux系统通常不允许用户使用root注册。安装后初次访问Ubuntu Linux系统时，只能使用安装系统时创建的用户名与密码注册，之后才能使用新增的用户名与密码注册。当然，经过一定的设置后，也可以使用超级用户root注册，详见第9章“用户管理”。

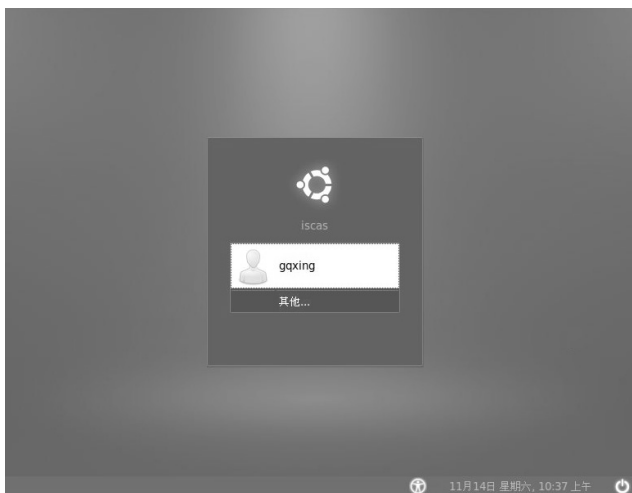


图1-20 系统注册界面

1.4.3 后期维护与更新

1. 修改grub.cfg配置文件

采用上述方法安装Ubuntu Linux系统之后，如果无法实现双引导，即无法正常启动Windows系统，就需要修改/boot/grub/grub.cfg配置文件，其中最正规的做法是运行update-grub或grub-mkconfig命令，重新生成grub.cfg配置文件，使之包含Windows系统引导菜单项：

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
[sudo] password for gqxing:
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-2.6.31-14-generic
Found initrd image: /boot/initrd.img-2.6.31-14-generic
Found memtest86+ image: /boot/memtest86+.bin
Found Microsoft Windows XP Home Edition on /dev/sda1
done
$
```

2. 安装附加的软件包

采用CD/DVD安装介质等方式安装的Ubuntu Linux系统只是一个基本系统，可能还无法完全满足用户的需求。为了安装其他附加的软件包，如Apache服务器、MySQL数据库以及Samba服务器等，可以采用apt-get、aptitude以及synaptic等软件管理工具。如果需要补充安装GNOME桌面环境支持的图形界面软件包，比较简单的方法是在GNOME桌面中选择“应用程序→Ubuntu软件中心”菜单，从中选择期望安装的软件包。有关软件的安装与维护，详见第3章“软件管理”。

3. 软件更新与系统升级

在提供系统支持期间，Ubuntu经常会发布各种更新软件包。更新软件包通常会修正软件中的错误或安全漏洞，提高软件的可靠性，或增加新的功能特性。为了确保系统的安全，应定期更新系统。

更新系统时, 可以采用`apt-get`、`aptitude`以及`synaptic`软件管理工具, 也可以从GNOME桌面中选择“系统→系统管理→更新软件包”, 或选择“系统→系统管理→新立得软件包管理器”菜单, 安装更新软件包。有关软件的安装、更新与维护, 详见第13章“软件管理”。

此外, Ubuntu Linux系统还会以后台进程的方式, 自动运行软件更新工具`update-notifier`与`update-manager` (其功能等同于“系统→系统管理→更新软件包”菜单), 检测系统配置的软件仓库中是否存在可用的软件包。当存在更新软件包时, GNOME窗口面板中将会出现一个后台运行的软件更新窗口, 提醒用户更新自己的系统, 详见第13章“软件管理”。

第2章 GNOME桌面

本章主要介绍Ubuntu Linux系统的GNOME桌面环境，简述其主要组成部分，以便读者对GNOME桌面环境有一个全面的了解。

在不同发行品牌或不同版本的Linux系统中，桌面环境之间的差别较大，但Linux系统的基本设计思想，尤其是作为底层支持的系统命令，除了新增功能之外，则很少有变化，这也是许多学习和使用Linux系统的人喜欢使用命令行界面的主要原因之一。因此，在学习Linux系统时，最好以熟悉命令行界面为主，把桌面环境作为一种辅助手段。但对于初学Linux系统的新用户而言，采用可视化的图形界面（而不是以命令方式）执行日常处理任务，可能相对比较容易一些。因此，初学者不妨先从熟悉GNOME/KDE桌面环境开始，逐步转入Linux系统命令行界面。

桌面环境是一个介于用户与操作系统之间的中间层，有助于用户完成诸如管理文件、运行程序等操作。为了适应Windows系统用户，Linux系统从纯粹的命令行界面逐步发展到提供GNOME（GNU Network Object Model Environment）和KDE（KDesktop Environment）等桌面环境。Ubuntu Linux系统的默认桌面环境是GNOME，但也支持KDE桌面环境，若想使用KDE桌面环境，可在Ubuntu Linux系统中安装KDE软件包，也可以安装一个纯KDE桌面环境的Kubuntu Linux系统。

2.1 GNOME桌面概述

2.1.1 GNOME注册界面

在成功地启动系统之后，控制台终端上将会出现一个Ubuntu Linux系统的GNOME注册界面，提示用户注册、访问Linux系统。每次注册时，即开启一个新的注册会话（从系统注册开始，直至从系统中注销的整个系统访问过程），如图2-1所示。

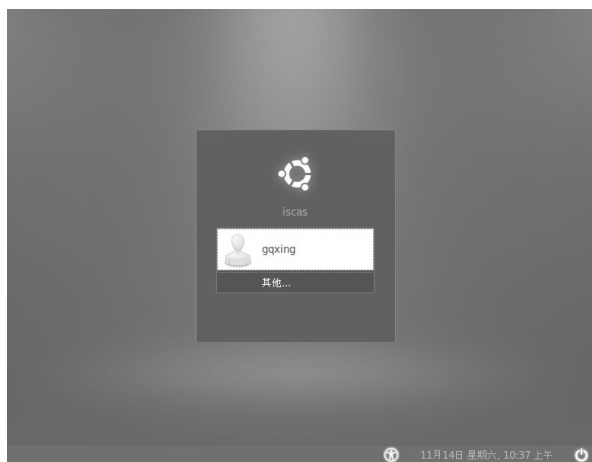


图2-1 Ubuntu Linux系统的注册界面

Ubuntu Linux系统的注册界面很简单，大体上可以分为两个不同的功能区域：注册区及信息与功能区。

注册区位于屏幕的正中位置，其中提供一个默认的注册用户名（以及其他可用的注册用户名），可以采用默认的用户名，也可以选用其他用户名，或单击“其他”栏目，从弹出的文本框中直接输入用户名，然后按Enter键。

信息与功能区位于屏幕的底部，用于显示系统的日期与时间，提供功能选择菜单。单击左边的人状按钮，也可以从弹出的菜单中选择“使用屏幕键盘”、“增强颜色的对比度”及“将文本放大以方便阅读”等人性化设置选项。单击右下角的电源开关按钮，还可以从弹出的菜单中选择暂时“休眠”、立即“重新启动”，或直接“关机”。

在输入用户名，按下Enter键之后，注册区将会增加一个密码输入文本框，如图2-2所示（由于虚拟机与笔记本电脑屏幕比例的原因，屏幕截图中左下角的“语言”二字未能正常显示）。

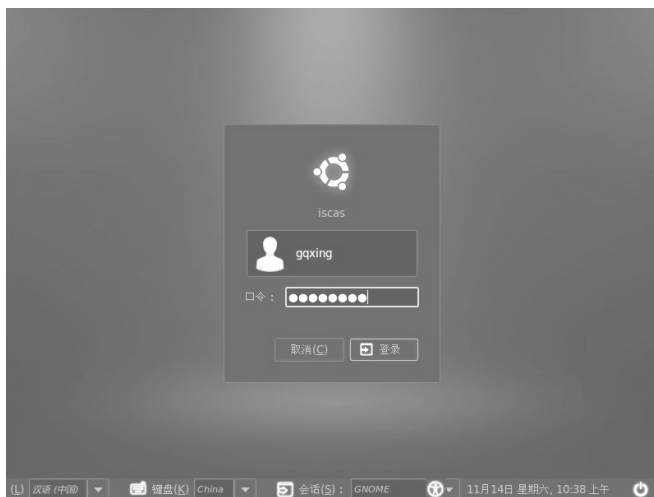


图2-2 密码输入界面

在此界面中，信息与功能区左边又增加了三个菜单项：“语言”用于选择系统会话期间使用的语言，如“汉语（中国）”、“汉语（台湾）”或“汉语（香港）”；“键盘”用于选择键盘类型，如China或USA等其他类型的键盘；“会话”用于选用GNOME、xterm以及安全模式GNOME。

在输入正确的密码之后，即可进入Linux系统，访问GNOME桌面环境。

2.1.2 GNOME桌面

Ubuntu GNOME桌面环境旨在提供一种可视化的图形界面，使用户能够通过图标、下拉菜单以及窗口等方式访问系统提供的处理功能。在GNOME注册界面中，如果输入的用户名与密码是正确的，即可成功地注册，进入Ubuntu Linux系统的GNOME桌面环境，如图2-3所示。

一旦注册到系统，GNOME桌面即成为用户的主要工作环境。通常情况下，大多数处理任务都可以在桌面环境中完成。Ubuntu GNOME桌面环境可以分为下列三个区域。

- 菜单面板；
- 桌面区；
- 窗口面板。



图2-3 Ubuntu Linux 系统的GNOME桌面

菜单面板位于GNOME桌面顶部的长条形区域，其中包含三个菜单和两个默认的应用启动图标，使用户能够快速地启动相应的应用程序。菜单面板的最右边是“信息公告区”，显示输入方法、网络控制、音量控制、当前注册的用户名、系统日期与时间，以及其他系统控制按钮与信息。

桌面区指的是菜单面板与窗口面板之间的整个屏幕区域。桌面区用于存放用户定义的应用程序启动器图标，存放安装的文件系统图标（如内置的Windows分区、CD/DVD或USB移动盘）等。

窗口面板位于屏幕的底部，其中最左边是一个“显示桌面”图标，紧接着可以是一系列表示用户当前打开的应用窗口的图标，右边存在2个工作区切换开关，最后是一个“回收站”图标。

屏幕顶部（或底部）的菜单面板可用于存放经常使用的项目，包括菜单、快速启动程序、按钮（图标）以及信息公告区等。Ubuntu GNOME桌面环境的布局和内容可以自己定制，用户可根据自己的需要，进行必要的调整，详见第2.6节。

2.2 GNOME桌面浏览

2.2.1 GNOME菜单面板

1. 主菜单

在GNOME桌面环境中，最常用的是菜单面板。菜单面板通常包含三个主要菜单：应用程序、位置和系统，用于访问Ubuntu Linux系统提供的各种功能。

- 应用程序——“应用程序”菜单包含各种菜单项，用于启动应用程序。GNOME把用户能够执行的程序分类组织成一级或二级菜单，如“Internet”、“办公”、“附件”、“图形”、“影音”、“游戏”以及“Ubuntu软件中心”等。“应用程序”菜单类似于Microsoft Windows系统的“开始”菜单。
- 位置——“位置”菜单用于调用文件浏览器，访问文件系统，包括主文件夹、定制的目录列表、安装的文件系统（如磁盘分区、USB移动盘以及CD/DVD等）、计算机以及网络等，也提供文件搜索功能及查询最近访问的文档等。

- 系统——“系统”菜单用于配置、管理与维护系统，其中包含“首选项（用于设置输入方法、窗口属性、鼠标、键盘与快捷键、默认打印机、屏幕保护程序、屏幕分辨率、桌面外观、网络代理、远程桌面、音响效果以及主菜单等）”、“系统管理（用于设置打印机、系统日期与时间、软件源、用户与用户组，查询系统日志，查询当前进程与系统资源状态，安装、更新及升级软件包等）”、“帮助和支持”以及GNOME与Ubuntu简介等。

GNOME桌面将许多常用的处理任务分类组织成三个菜单。用户可以从三个主要菜单开始，按照菜单的级联层次，逐层向下检索各级子菜单的菜单项，直到找到自己需要的处理任务，然后选择执行任何一项具体的处理任务。如果需要，也可以采用拖放形式，在桌面中创建菜单项的快捷键。安装系统之后，初始的菜单面板如图2-4所示。



图2-4 菜单面板

2. 快速启动区

除了三个菜单选项之外，菜单面板的中间部分称做快速启动区，其中包含下列两个默认的通用软件图标（或称按钮），能够快速启动相应的应用程序：

- Firefox网络浏览器；
- Ubuntu帮助信息。

除了上述两个基本图标之外，如有必要，可以把一部分频繁调用的程序，以图标形式置于快速启动区，单击相应的图标，即可绕过菜单，快速启动相应的程序。如果桌面区存在已经创建的应用程序启动器，也可以选中启动器图标，将其拖放至快速启动区。

如果想要在快速启动区中增加其他应用程序图标，可以右击菜单面板的空白区域，从上下文菜单中选择“添加到面板”菜单项，然后从弹出的“添加到面板”窗口中选择期望增加的处理功能，最后单击“添加”按钮，参见图2-5。

在GNOME菜单面板中，如果右击任何一个菜单或快速启动图标，将会显示其上下文菜单，列出相应的功能选项，以及能够执行的各项处理任务。

3. 信息公告区

菜单面板的右边（屏幕右上角）是信息公告区。在Ubuntu 9.10 Linux系统的信息公告区中，从右到左依次为用户显示与关机、日期与时间、即时消息与电子邮件、音量控制、语言与输入方法、电源指示、网络连接及蓝牙设备安装向导等图标。注意，由于系统的并发启动特性，系统每次启动后的图标摆放顺序并不唯一，而且，如果系统的配置不同，图标的种类与数量也不一定相同。

上述图标通常是由Applet实现的，能够不断更新，以提醒用户注意。通常，只需单击图标即可启动相应的程序，也可以右击图标，从菜单中选择适当的功能。



图2-5 菜单面板添加图标

“用户显示与关机”图标具有综合性的功能，通常用于显示当前注册的用户，单击这一图标将会弹出一个菜单，可以选择“切换用户”、“锁定屏幕”、“注销”、“挂起”、“休眠”、“重启”和“关机”等菜单项。

“切换用户”菜单用于切换用户，以便能够以不同的用户身份访问系统。

顾名思义，“锁定屏幕”用于锁住控制台桌面，启动屏幕保护程序，防止他人访问Ubuntu Linux系统。当需要继续工作时，可以按任何键，但只有正确输入自己的密码之后才能恢复桌面会话。

“注销”功能用于注销当前的用户，结束系统会话，退出Ubuntu Linux系统。如果计算机是共享的，注销之后可使其他用户继续注册访问Linux系统。选择“退出”菜单之后，系统将会弹出一个窗口，同时给予用户60秒改变决定的时间。用户可以单击“退出”按钮立即注销，或单击“取消”按钮返回。也可以等待超时后退出GNOME桌面环境（默认选择）。选择“注销”后，系统将会返回注册界面，此时可用其他用户名再次注册到系统。

当选择“关机”或“重启”菜单时，系统将会弹出一个窗口，同时给予用户60秒改变决定的时间。用户可以选择继续关机或重新启动系统，也可以等待超时后由系统自动关机或重新启动。如果不想关机或重启系统，可以单击“取消”按钮。一旦关机，系统即可切断计算机电源。选择“挂起”或“休眠”菜单将会立即进入临时关机状态。一旦需要再次访问系统，按下电源开关，即可进入先前的运行环境继续工作。

“日期与时间”图标用于显示系统的日期与时间。单击“日期与时间”图标将会弹出当月的日历。再次单击“日期与时间”图标将会取消日期和时间显示。右击“日期与时间”图标时将会弹出一个上下文菜单，其中，“首选项”菜单项可用于修改显示的内容，以及日期与时间的显示形式，单击“时间设置”按钮允许特权用户调整系统的日期与时间等。

“信封”图标用于快速启动Empathy即时消息或Evolution电子邮件。

“音量控制”图标用于调整音响的音量。右击“音量控制”图标，可以设置音响的静音等。

“键盘”图标用于选择输入方法，如英文或简体中文。右击小键盘图标，从弹出的上下文菜单中选择“首选项”，可以查询、选择或增加输入方法，也可以选择候选词的排列方向。之后，图标将会变为表示相应输入方法的一个汉字，如“拼”表示“拼音”等。无论何时（如运行Office办公软件、文本编辑器，以及打开终端窗口时），如果需要输入中文，可以单击菜单面板信息公告区中的键盘图标，从弹出的菜单中选择中文输入方法。

当需要输入中文时，也可以同时按下Ctrl与空格组合键，桌面右下角将会出现一个中文输入方法条形框，此时即可输入中文。默认的输入方法为“拼音”。此外，同时按下Ctrl与空格键，还可以在中英文输入方法之间切换。

“电源状态”图标（如果存在）是一个电源状态指示器，用于显示电源的剩余电量。

“网络”图标用于查询网络的连接状态，以及选择、连接可用的网络，如有线网络与无线网络等。

4. 面板上下文菜单

在菜单面板（包括窗口面板）的空白处单击鼠标右键，可以弹出一个与菜单面板有关的上下文菜单，通过选择其中的菜单项，可以在面板中增加应用程序的快速启动图标，从而能够快速调用相应的处理程序。此外还可以设置面板的属性，如面板的方向与浮动等，如图2-6所示。



图2-6 面板上下文菜单

表2-1 列出了菜单面板上下文菜单提供的菜单项。

表2-1 菜单面板的上下文菜单

菜单项	基本功能
添加到面板	在菜单面板上创建应用程序快速启动图标，如“搜索文件”、“网络监视器”、“系统监视器”、“Eyes（眼睛）”与“气象报告”等
属性	定义菜单面板的属性，如面板的方向与大小（像素），以及是否自动隐藏等
删除该面板	删除菜单面板
新建面板	创建一个新的面板，如在桌面的左右创建新的面板，以丰富自己的桌面环境等
帮助	介绍面板的基本概念与使用方法，如菜单、按钮（图标）与快速启动的简单说明与用法
关于面板	说明GNOME面板程序的版本等

2.2.2 GNOME桌面区

1. 图标

GNOME桌面区是一个存储空间，也是用户主目录下的一个子目录，可用于存放启动器图标、文件或目录等。最初，GNOME桌面区是空的。当插入移动存储介质，如CD/DVD或USB移动盘时，GNOME桌面区中将会自动出现表示相应介质的图标。

此外，如果计算机中还装有Windows系统，从“位置”菜单选择磁盘卷标（如A22eXP）等菜单项，桌面区也会出现表示Windows系统所在磁盘分区的图标，说明已经把Windows文件系统自动安装到Linux系统的/media目录中。当安装了Windows文件系统，接着又插入一个1GB的U盘时，其结果如图2-7所示。



图2-7 桌面区图标

在GNOME桌面中，双击任何一个图标即可激活图标表示的功能。如果右击图标，将会显示图标的上下文菜单，列出相应的功能选项，以及能够执行的各项处理任务。

2. 桌面区上下文菜单

在GNOME桌面的空白处单击鼠标右键，可以弹出一个与桌面区有关的上下文菜单，通过选择其中的菜单项，可以执行相应的处理任务，如图2-8所示。

表2-2列出了桌面区上下文菜单提供的菜单项。

表2-2 桌面区上下文菜单

菜单项	基本功能
创建文件夹	在桌面上创建一个“未命名文件夹”（可以重新命名）
创建启动器	弹出一个“创建启动器”对话框，以便在桌面上创建一个能够快速启动的应用启动器图标
创建文档	在桌面上创建一个名为“新文件”的空文件（可以重新命名）
按名称清理	按照字母顺序排序桌面上的图标
保持对齐	按栅格形式整齐排列桌面上的图标
粘贴	用于粘贴先前复制的数据副本
更改桌面背景	弹出一个能够修改桌面背景的对话框，使用户能够选择不同的图像或照片，作为桌面背景

除了菜单面板之外，也可以把桌面区作为一个快速启动区，创建应用程序启动器，把应用程序快速启动图标置于桌面区。双击桌面区存放的图标，即可快速启动相应的应用程序。

当从桌面区上下文菜单中选择“创建启动器”时，将会弹出一个“创建启动器”对话框，可用它创建一个快速启动器，如图2-9所示。



图2-8 桌面区上下文菜单



图2-9 “创建启动器”对话框

从“类型”下拉菜单中选择启动器的类型（如“应用程序”），在“名称”栏目中输入启动器图标的名字（如“文件浏览器”），在“命令”栏目中输入或单击“浏览”按钮选择实际运行的命令（如“nautilus”），如果需要，也可以在“注释”栏目中给出简单的注释信息，最后单击“确定”按钮，即可在桌面区创建一个启动器图标。单击这个图标可以快速启动文件浏览器，如图2-10所示。

如果选择的启动器类型是“应用程序”，每当双击启动器图标时，即可快速地启动相应的应用程序。如果选择的是“终端中的应用程序”，双击启动器图标时将会打开一个新的终端窗口，并在其中运行指定的命令。如果选择的启动器类型是“位置”，且命令字段给定的是一个文本文件，双击启动器图标时将会调用文本编辑器，打开相应的文件。如果命令字段给定的是

目录而不是文件，GNOME将会调用文件浏览器，打开相应的目录。如果命令字段给定的是脚本文件，GNOME将会询问用户在终端窗口中运行，还是显示文件的内容。

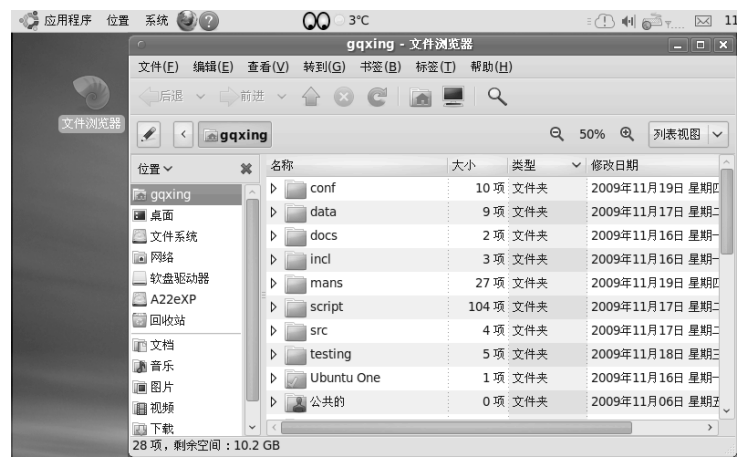


图2-10 文件浏览器启动器图标及其应用

2.2.3 GNOME窗口面板

GNOME桌面底部的长条形区域称做窗口面板。窗口面板的主要作用是显示当前打开的应用窗口，切换桌面工作区，切换当前打开的应用窗口，以及显示或隐藏GNOME桌面区等。窗口面板通常包含下列四个组件（如图2-11所示）。

- 桌面显示/隐藏按钮;
- 窗口列表;
- 工作区切换开关;
- 回收站图标。

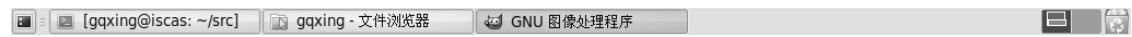


图2-11 GNOME窗口面板

屏幕左下角的“桌面显示/隐藏”按钮用于展示整个桌面区，隐藏桌面上已经打开的所有活动窗口。当桌面上打开大量的窗口时，这是一项很有用的功能。再次单击“桌面显示/隐藏”按钮时，又可恢复原来打开的窗口。

当前打开的窗口列表位于窗口面板的中部。打开任何一个应用程序时，除了在桌面上显示一个活动的窗口外，还会在窗口面板中部的“窗口列表”中显示一个窗口图标。同Windows系统类似，单击任何一个应用窗口图标，即可激活相应的应用窗口，并将其置于所有窗口的最上面。如果单击窗口右上角的“最小化”按钮，窗口将会从桌面上消失，但“窗口列表”中的窗口图标继续存在。单击“窗口列表”中的窗口图标，可以在其他所有窗口的上层恢复窗口显示，使窗口处于活动状态。另外，也可以使用Alt+Tab组合键，切换至已经打开的任何应用窗口。

工作区切换开关位于窗口面板的右边。每个工作区都拥有一个单独的GNOME桌面，包括菜单面板、桌面区和窗口面板等。所有工作区的菜单面板和背景主题都是相同的。在每个工作区中，可以运行不同的应用程序。工作区切换开关使用户能够在工作区之间相互切换。

回收站图标位于窗口面板的最右边，用于缓存桌面环境中删除的文件等。

1. 窗口列表

窗口列表（或称任务栏）用于展示当前打开的所有窗口，使用户能够随时切换活动的窗口。GNOME允许同时打开多个并发运行的窗口。窗口列表中将会根据打开的顺序，依次排列当前运行的每一个窗口。当单击窗口列表中的任何一个窗口标识时，选中的窗口就会变成活动的窗口。但如果单击的是一个活动的窗口，窗口将会消失。

2. 虚拟工作区

GNOME桌面提供两个虚拟工作区。除了当前使用的GNOME桌面，还有1个虚拟工作区桌面可以随时使用。当需要在其他虚拟工作区桌面中运行不同的程序时，可以单击工作区切换开关，直接切换到相应的虚拟GNOME桌面。当需要在不同的虚拟工作区桌面并发地运行不同的应用程序时，工作区切换开关能够使GNOME工作区桌面的切换变得简单快捷。

3. 回收站

回收站用于缓存删除的文件等。在桌面环境中，用户删除的任何文件、目录以及图标等将会自动移至回收站（命令行中删除的文件不在此列）。若想永久性地删除文件，或定时清理回收站，可以右击“回收站”图标，在弹出的“回收站 - 文件浏览器”窗口中，选择删除选定的文件，或清空回收站。如果想在删除文件时能够绕过回收站，彻底删除文件，可在删除文件时按住Shift键，然后再删除文件。注意，在删除大量文件之后，如果不及时清理回收站，将会挤占可用的存储空间。



图2-12 应用程序主菜单

2.3 应用程序菜单

在GNOME桌面环境中，常规的处理功能均已纳入“应用程序”主菜单中。单击“应用程序”菜单时，将会分类列出各种内置实用程序的子菜单。单击任何一个子菜单将会列出一组可用的实用程序（或下一级子菜单），如图2-12所示。注意，如果系统里也安装了KDE桌面环境，KDE的内置应用程序和实用程序也会列在子菜单中，因而也可以在GNOME桌面环境中访问。

应用程序菜单通常包含Internet、办公、附件、图形、影音、游戏以及Ubuntu软件中心等子菜单。下面分别介绍其中的部分主要应用程序。

2.3.1 Internet

“Internet”菜单主要提供网络浏览器、电子邮件、即时消息、Ubuntu One网络存储以及BT下载客户端等功能。

1. Firefox浏览器

Firefox是GNOME桌面环境提供的默认Web浏览器。Firefox具有丰富且安全的浏览功能。安装Ubuntu Linux系统之后，即可使用Firefox浏览器访问Internet，浏览网页，复制网页内容，保存与打印网页，以及下载文件等，如图2-13所示。如果需要，也可以在Firefox中选择“编辑→首选项”菜单，定制Firefox中的设置选项，如设置主页及下载网页的存放位置等。Firefox是一

种快速的小型浏览器，其插件功能使Firefox能够根据需要随时扩展，以支持新的功能特性，其具体特性读者可以参见<http://www.mozilla.org/support/firefox>。



图2-13 Firefox网络浏览器

2. Evolution 电子邮件

GNOME利用Evolution作为通信工具，提供电子邮件、日历、计划安排及信息组织等功能，使用户之间能够相互传递信息，如图2-14所示。



图2-14 Evolution电子邮件主界面

Evolution能够连接许多电子邮件服务器，如gmail、yahoo和Exchang等，查询或收发电子邮件。下面以gmail邮件服务器为例，说明如何设置。

Evolution可以在第一次启动时设置，也可以在随后的运行过程中设置或修改，两者的设置内容基本上是相同的。如果初次运行时设置得不正确，可以选择“Internet→Evolution邮件”菜单，在进入Evolution电子邮件界面之后选择“编辑→首选项”菜单，根据自己的用户账号是否存在，单击“Evolution首选项”界面中的“添加”或“编辑”按钮，在“账户编辑器”界面中，

首先设置“标识”标签页：在“名称”栏目中输入账户名称，在“全名”栏目中输入自己的正式名字，在“电子邮件地址”栏目中输入自己的gmail电子邮件地址，如图2-15所示。

在“接收电子邮件”标签中，从“服务器类型”下拉框中选择“POP”，在“服务器”栏目中输入“pop.gmail.com”，在“用户名”栏目中输入自己的名字或gmail电子邮件地址。从“使用安全连接”下拉框中选择“SSL加密”，在“认证类型”下拉框中选择“密码”，最后勾选“记住密码”复选框，如图2-16所示。



图2-15 账户编辑器（1）



图2-16 账户编辑器（2）



图2-17 账户编辑器（3）

在“发送电子邮件”标签中，从“服务器类型”下拉框中选择“SMTP”，在“服务器”栏目中输入“smtp.gmail.com”，勾选“服务器需要认证”复选框，从“使用安全连接”下拉框中选择“SSL加密”，在身份验证“类型”下拉框中选择“PLAIN”，在“用户名”栏目中输入自己的gmail电子邮件地址，最后勾选“记住密码”复选框，如图2-17所示。

至此，Evolution电子邮件连接gmail服务器的设置基本上已经完成，“接收选项”、“默认”和“安全”等标签页暂时可以不考虑。为了验证刚才的设置，可以单击Evolution电子邮件主界面中的“发送/接收”按钮，如果出现输入密码对话框，表示设置已经成功，可以收发电子邮件了。

如果测试不成功，可以利用Firefox浏览器连接gmail服务器，单击gmail主界面右上方的“settings（设置）”标签，在“settings”页面中单击上方的“Forwarding and POP/IMAP（转发和PCP/IMAP）”标签，确保“PCP Download（PCP下载）”栏目中的“Enable PCP for all mail（针对所有邮件启用POP）”单选框已选择。如有修改，单击“Save Changes（保存更改）”按钮，保存新的设置。

3. Ubuntu One

Ubuntu One是一种网络存储服务或文件共享服务，可用于存储、共享和同步文件，实现

个人的云计算等。Ubuntu One能够使用户脱机存储自己的基本信息, 备份图片、视频和音乐等, 在用户使用的任何系统与服务器之间自动实现数据的同步, 达到数据共享的目的。若想访问Ubuntu One, 存储数据文件, 可以选择“Internet→Ubuntu One”或“位置→Ubuntu One”菜单, 存储的文件将会自动同步到Ubuntu远程存储服务器。若想直接访问Ubuntu One服务器, 可以右击信息公告区中的云形图标, 从弹出的菜单中选择“访问网页”, 注册后即可进入自己的账号, 如图2-18所示。

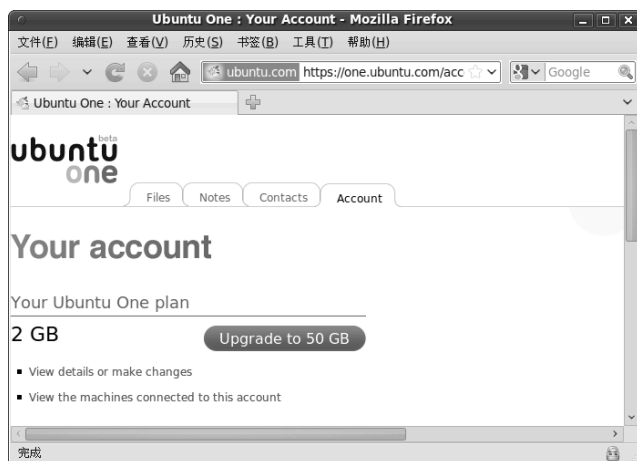


图2-18 Ubuntu One

利用Ubuntu One实现远程文件共享是一种理想的方法, 能够快速共享任何文件, 且文件的大小不限。Ubuntu提供2 GB的免费存储空间, 必要时可以申请高达50 GB的存储空间, 但每月需付10美元的维护费。在使用Ubuntu One之前, 需用电子邮件地址以及launchpad账号与密码进行注册。一旦注册成功, 就可以在任何Ubuntu Linux系统中访问Ubuntu One了。

4. Empathy即时消息

即时消息 (Instant Messaging, IM) 是目前常用的一种通信方式。利用短消息与同事或朋友联系, 有时比电子邮件或电话更方便快捷。Ubuntu 9.10 Linux系统以Empathy取代了先前的Pidgin。Empathy IM客户端软件支持Jabber、Google Talk、AIM、ICQ、MSN、QQ和Yahoo!等通信协议。

2.3.2 办公

GNOME提供一整套免费的办公应用程序OpenOffice.org, 使用户能够创建基于开放标准格式的文档、电子表格以及演示文稿等, 如OpenDocument、Rich Text和HTML等。如有必要, 也可以读写和编辑Microsoft Office Word、Excel和PowerPoint等格式的文档, 且具有高度的兼容性。此外, GNOME还提供字典、Evolution邮件和日历, 以及项目管理等工具。

1. OpenOffice.org

OpenOffice.org的功能类似于Windows Office。其中最常用的三个OpenOffice应用是文字处理 (Writer)、演示文稿 (Impress) 及电子表格 (Calc)。

- Writer类似于Microsoft Office Word, 可用于创建、编写文档、文本和网页。但不同的是, Writer能够直接创建PDF文件。

- Impress类似于Microsoft Office PowerPoint，可用于创建、编写演示文稿以及幻灯片。同Writer一样，Impress也能够直接创建PDF文件。
- Calc类似于Microsoft Office Excel，可用于创建、编写电子表格，计算、分析数据信息。Calc也可以直接生成PDF文档。

同Microsoft Office一样，在任何一个OpenOffice.org应用中建立的对象可以导入另一个应用中。例如，电子表单可以导入Writer文档，或作为Impress幻灯片的一个组成部分。同样，Impress中的图像也可以导入Writer文档。此外，由于OpenOffice.org的文件都是与处理文件的应用相关联的，故双击文件也可以启动相应的应用程序，打开选定的文件。

OpenOffice.org办公套件的一个重要特点或功能是能够读取并处理Microsoft Office格式的文件，把Microsoft Office格式的文件转换成Open Document格式的文件。例如，如果在OpenOffice.org中选择“文件→打开”，然后在对话框中选择any.doc文件，将会打开在Microsoft Office Word中创建的any.doc文档。

除了Writer、Impress与Calc之外，OpenOffice还提供Base数据库和Math公式编辑等应用程序。Base是一个类似于Microsoft Access的数据库，可用于建立关系数据库，实现基于数据库的应用。Base也可与其他OpenOffice.org程序交换数据，如利用数据库中的数据生成电子表单等。Math公式编辑应用程序提供数学公式的编辑与计算功能，既可单独使用，也可作为一种对象，用于Writer与Calc等。

限于篇幅，本章不打算详细介绍上述应用程序的具体操作，如想了解OpenOffice.org的更多信息，可以访问<http://www.openoffice.org>等网站。

2. 字典

GNOME采用默认的网络字典(dict.org)提供字典服务。用户可以查询英文单词的释义与用法，GNOME将会在显示区内显示检索结果。如果单词拼写错误，GNOME会启动拼写检查程序，提供一个列表供用户选择拼写正确的单词，然后再查询。此外，用户也可以在输出显示区内检索指定的关键字。字典查询的结果能够复制到GNOME桌面的其他应用中，也可以保存或打印。注意，GNOME不使用本地字典，必须连接到Internet才能使用。除了默认的网络字典，用户也可以指定其他提供字典服务的网站作为字典服务器，如图2-19所示。

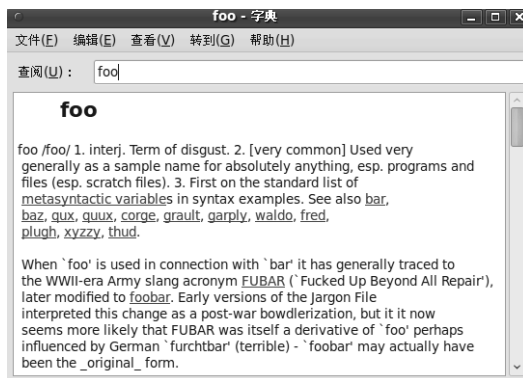


图2-19 字典

此外，GNOME还提供一个功能丰富的项目管理工具Planner，能够帮助用户管理与跟踪项目的任务分配、完成日期以及所用的资源。可以按阶段分配任务，为任务赋予重要性级别，必

要时甚至重新分配任务等。用户可以利用可视化的Gantt图表显示项目的进度，了解项目的进展情况，采用高亮度方式标注项目阶段需要完成的关键任务等。



注意

Planner软件包需要单独安装，有关Planner项目管理工具的更多信息，可以访问<http://live.gnome.org/Planner>网站。

2.3.3 附件

GNOME提供一组标准的附件，便于用户执行基本的处理功能。单击“应用程序→附件”菜单，即可选用终端仿真、磁盘使用分析、多功能计算器、gedit文本编辑器、屏幕抓图以及CD/DVD刻录工具等。

1. 终端

“终端”提供命令行终端窗口，以便用户能够采用命令行方式，直接访问Shell和Linux系统。用户可以修改终端窗口的背景图案（包括透明背景），设置文字和背景的颜色，以及设置窗口滚动缓冲区等。此外，终端窗口也支持标签操作等。这是许多资深Linux用户最喜欢使用的一种用户界面，后续各章介绍的Ubuntu Linux系统命令行功能都需要用到此界面，参见第3章“命令行基础知识”。

2. 文本编辑器

文本编辑器gedit相当于Windows系统中的记事本功能，可用于创建和编辑简单的文本文件。文本编辑器可以同时打开和处理多个文件，每个文件位于一个单独的窗口。如果需要，可以从一个文件中复制、剪切文本数据，然后粘贴到另外一个文件中。

此外，gedit文本编辑器还可以用做软件开发工具。当通过“查看→突出显示模式”菜单选择不同的语言时，如C/C++、Java、HTML、XML、Shell、Perl或Python等，gedit文本编辑器会根据不同的语言，以不同的颜色或高亮度方式显示其中的关键字和标识符等，如图2-20所示。

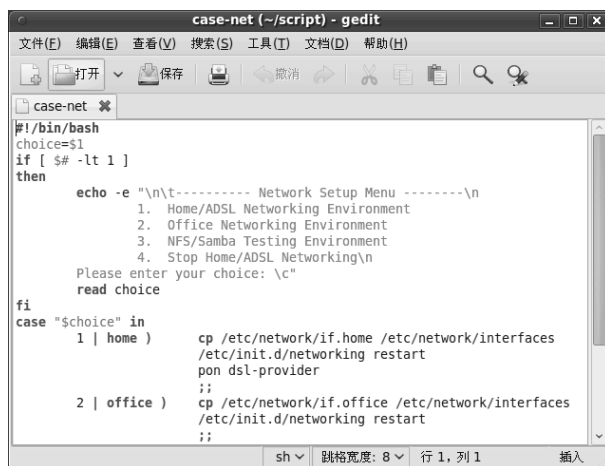


图2-20 gedit文本编辑器

3. 磁盘使用分析器

“磁盘使用分析器”是一个图形界面的磁盘维护工具，用于分析磁盘的使用情况。“磁盘使用分析器”能够扫描整个文件系统、用户主目录或指定的任何目录分支等，最后以分类统计

的报表形式或以形象化的磁盘空间占用形式，给出磁盘使用情况的分析结果，如图2-21所示。

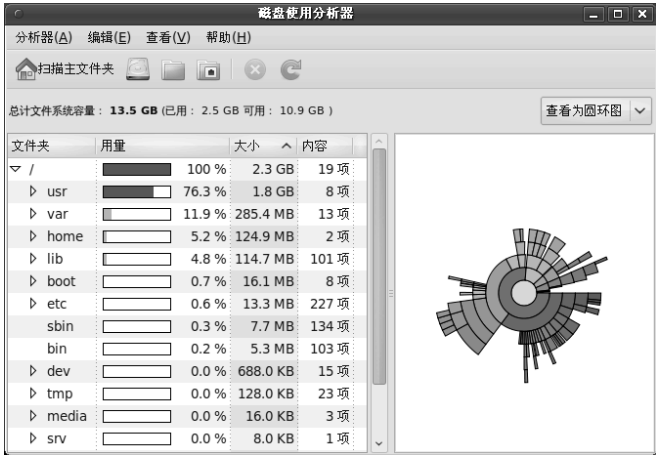


图2-21 磁盘使用分析器

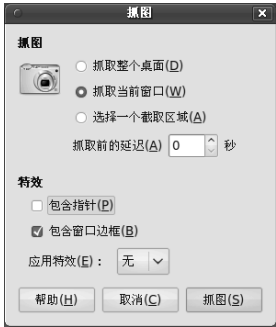


图2-22 抓图

4. 抓图

“抓图”用于捕捉和保存屏幕快照，如抓取整个桌面，抓取当前窗口。此外还可以设定抓图的延迟时间，这对于抓取屏幕快照是非常有用的。本书采用的部分屏幕截图也是利用“抓图”功能制作的，如图2-22所示。

5. CD/DVD创建器

“CD/DVD创建器”用于刻录CD/DVD，如图2-23所示。如果想把数据文件或文件夹写入CD/DVD，或制作CD/DVD映像文件，可以利用GNOME桌面提供的“CD/DVD创建器”功能，把空白CD/DVD插入刻录机之后，仿照下列步骤执行。

- (1) 选择“应用程序→附件→CD/DVD创建器”菜单。
- (2) 进入“CD/DVD生成器-文件浏览器”窗口。



图2-23 CD/DVD创建器界面

(3) 打开另外一个文件浏览器, 找出并选择准备复制的源文件, 把文件拖放至“CD/DVD 创建器文件夹”中的空白处。

(4) 单击右上角的“写入到盘片”按钮。

(5) 必要时, 也可以在弹出的“光盘刻录设置”对话框中修改CD/DVD的卷标名称等;

(6) 单击“光盘刻录设置”对话框右下角的“刻录”按钮。

2.3.4 图形

GNOME桌面提供了若干图形图像应用程序, 每一种图形图像应用程序都有其特定的用途, 如F-Spot照片管理器、GIMP图片编辑器、OpenOffice.org图画以及XSane图像扫描器等。

1. F-Spot照片管理器

F-Spot照片管理器能够分类组织、浏览、检索、旋转、缩放以及打印数字图像文件, 能够编辑、标记、剪裁图像, 调整色彩, 如调整色调、亮度、饱和度及对比度等, 以增强图像效果, 也能够以幻灯片形式播放图像文件, 如图2-24所示。



图2-24 F-Spot图片管理器

2. GIMP图片编辑器

GIMP (GNU Image Manipulation Program) 是一个功能强大的图像处理器, 提供一整套的画笔、铅笔和喷枪等编辑工具, 可用于处理数字图像和照片, 如创建、缩放、剪辑、改变颜色、分层组合图像、删除部分图像特征以及在不同的图像格式之间转换等。GIMP还可用于生成简单的动画图像, 也可用于捕捉、制作和保存屏幕快照。实际上, 本书采用的部分屏幕截图也是利用GIMP生成的。欲想了解怎样使用GIMP, 可以访问<http://wikiubuntu.org.cn>网站, 其“站点导航”的“影音图像”中提供了一个完整的用户手册。图2-25是GIMP图片编辑器的主要界面。

2.3.5 影音

Ubuntu GNOME提供部分音频与视频播放工具, 用于欣赏音乐和播放影视, 其中包括光盘刻录器、Rhythmbox音乐播放器、电影播放机及录音机等。为了访问这些多媒体工具, 可以选择“应用程序→影音”菜单。

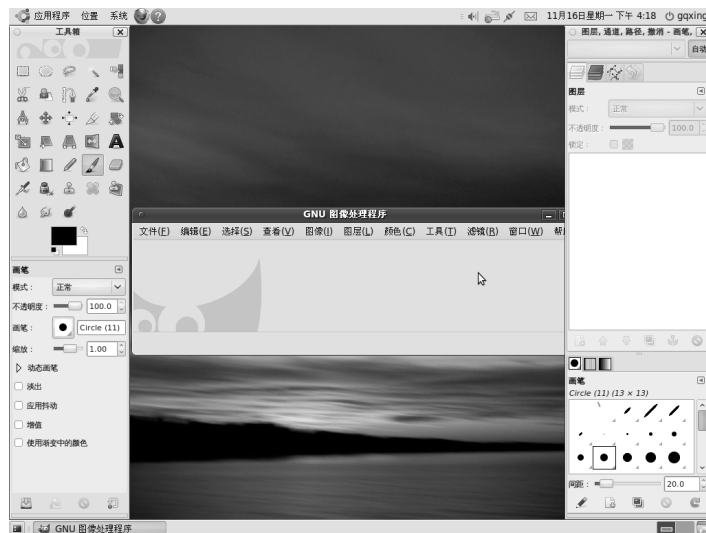


图2-25 GIMP图像工具

1. Brasero光盘刻录器

Brasero是一个CD/DVD刻录工具，支持音频CD、视频DVD/VCD和数据CD/DVD等刻录模式，可用于录制CD/DVD，利用ISO镜像刻录CD/DVD，制作光盘数据文件备份，检测光盘文件的完整性，擦除可重写的光盘，支持光盘的封面设计等。Brasero光盘刻录器的用户界面直观简洁，操作简便，如图2-26所示。



图2-26 Brasero光盘刻录器

2. Rhythm box音乐播放器

Rhythm box是一个音乐管理与播放软件，配有音乐店Jamendo和Magnatune，支持一系列网络电台，能够组织多媒体文件，如音乐、CD和网络电台等，构建自己的多媒体库，播放音乐文件，从音频CD上导入音乐文件，聆听Internet网络电台等。此外，Rhythm box还可以作为首选的MP3音乐播放器，如图2-27所示。

3. 电影播放机

Totem电影播放机(Totem Movie Player)是一个全功能的电影播放器，支持视频和音频文件，能够播放DVD/VCD影视，播放音频CD，具有屏幕影像控制以及键盘控制功能，能够像

DVD机一样控制影视的播放（必要时可以补充安装gststreamer0.10-plugins-bad及gststreamer0.10-plugins-ugly等音频和视频插件软件包），如图2-28所示。



图2-27 Rhythmbox音乐播放器



图2-28 Totem电影播放机

此外还可以选用MPlayer、SMPlayer和RealPlayer等视频播放器，但需要单独安装，详见<http://wikiubuntu.org.cn>。其中，MPlayer能够播放MPEG/VOB、AVI、Ogg/OGM、VIVO、ASF/WMA/WMV、QT/MOV/MP4、RealMedia、Matroska、NUT、NuppelVideo、FLI、YUV4MPEG、FILM、RoQ、PVA等格式的视频文件，也可用于播放VideoCD、SVCD、DVD、DivX3/4/5、WMV甚至H264等格式的电影。

4. 录音机

Sound Recorder录音机用于录制.flac、.oga以及wav格式的音频文件，能够利用计算机系统配置的扬声器或耳机播放音乐，如图2-29所示。



图2-29 录音机

2.3.6 游戏

通常，GNOME至少提供10个游戏，如Gnome方块、对对碰以及国际象棋等。安装之后，还可以利用“应用程序→Ubuntu软件中心”菜单，从Ubuntu软件仓库提供的大量游戏软件中选择下载自己喜欢的游戏。

2.3.7 Ubuntu软件中心

用于下载、安装或删除软件包。详见第13章“软件管理”。

2.4 位置菜单

“位置”菜单的主要功能是快速定位系统资源，如快速进入用户主目录等。“位置”菜单提供的大部分功能均采用nautilus文件浏览器供用户访问文件系统，浏览系统配置的各种外部设备，检索可用的网络资源，以及检索目录与文件等，如图2-30所示。

Nautilus是GNOME内置的文件管理器，使用户能够浏览文件和文件夹，其中包括：

- 管理文件和文件夹，定制文件夹的显示形式；
- 将数据写入移动存储介质，如USB移动盘和CD/DVD等。

Nautilus文件浏览器的功能类似于Windows系统中的资源管理器。其中，左边的“位置”窗口中列出了可以浏览的目录、文件系统或存储介质。右边的主窗口中列出了目录、文件系统或存储介质中包含的文件或文件夹。利用文件浏览器，可以浏览、复制、移动、创建、删除以及修改文件与文件夹，可以查阅文件和文件夹的内容，也可以按照指定的模式检索指定的文件与文件夹。

在Ubuntu Linux系统中，Nautilus文件浏览器还提供标签支持，可以通过标签浏览不同目录、文件系统或存储介质中的文件或文件夹。

Nautilus文件浏览器能够以图标形式显示文件和文件夹，能够以详细列表的形式显示文件和文件夹，同时给出文件的大小、创建日期及类型等信息，也能够以紧凑方式显示文件和文件夹。图2-31就是以紧凑方式显示“/”文件系统的结果。



图2-30 “位置”菜单

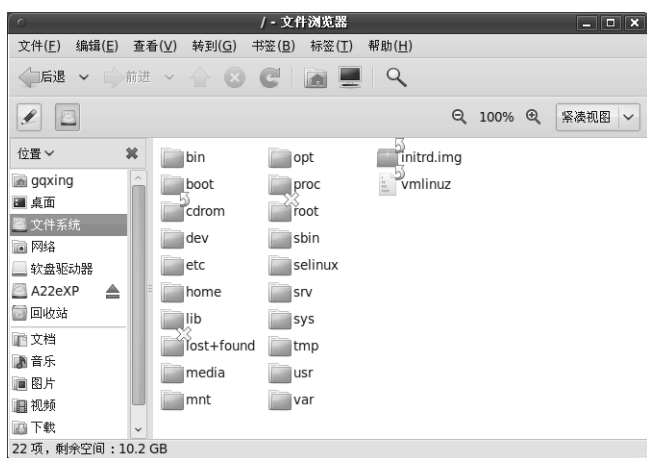


图2-31 文件浏览器

Nautilus文件浏览器采用不同的图标区别文件和文件夹，区分不同类型的文件。例如，以公文包图标表示文件夹；以计算机芯片图标表示可执行程序文件；以含有计算机芯片的图标表示脚本文件；以纸张加文字的图标表示文本文件；以图像图标表示图像文件；以盒状图标表示压缩文件；以文档、电子表单和幻灯片等图标表示OpenOffice.org文件；以含有数字0的图标表示系统文件等。当以图标形式显示文件夹或文件时，在图标的下方（或右边）将会显示文件夹或文件的名字，如图2-32所示。



图2-32 文件浏览器图标显示

在文件浏览器中，有些文件或文件夹上还存在附加的标记，用于表示文件或文件夹的访问权限，以及符号链接文件。例如，锁状附加标记表示当前的用户只能读但不能写相应的文件或文件夹。叉号“×”附加标记表示当前用户根本就不能读写相应的文件或文件夹。含有箭头附加标记的图标形式表示符号链接文件等。

在文件浏览器中，利用鼠标单击文件，可以打开或运行选中的文件。与Windows系统一样，文件的类型或扩展名可以决定与文件相关联的默认操作。双击一个可执行文件，将会启动相应的程序（假定用户拥有执行文件的访问权限）；双击一个文本文件，系统就会利用gedit文本编辑器打开文件（假定用户拥有读文件的访问权限）。用鼠标拖放文件或文件夹时，可以把文件或文件夹从一个文件夹移至另一个文件夹或文件系统，前提同样是需要拥有适当的访问权限。

2.4.1 主文件夹

“主文件夹”是当前注册用户的主目录，相当于Windows系统的“我的文档”，对于普通用户而言，主文件夹即/home/username目录，其中的“username”是当前注册的用户名。

单击“主文件夹”菜单即可进入用户的主目录，利用文件浏览器浏览和访问其中的文件与目录，如图2-33所示。主文件夹用于存储用户自己的所有文件。对于不同的用户而言，其主文件夹是不同的。

2.4.2 桌面、文档等

“桌面”、“文档”、“音乐”、“图片”、“视频”及“下载”是用户主目录下的子目录，选择任何一个菜单项，即可调用文件浏览器，快速访问其中的文件。



图2-33 主文件夹

2.4.3 计算机

“计算机”用于访问计算机系统配置的所有硬件存储设备，包括已安装的移动介质，如“软盘驱动器”、“CD/DVD驱动器”、Windows系统分区（如A22eXP）、位于系统磁盘上的“文件系统”和USB移动盘等，其功能等价于Microsoft Windows系统的“我的电脑”，可用于访问其中存储的文件等，如图2-34所示。



图2-34 “计算机”界面

如果插入CD/DVD或USB移动盘，系统将会自动安装相应的文件系统。否则，可以直接单击相应的图标，安装相应的文件系统（其安装点位于/media目录下）。

2.4.4 磁盘分区

如果系统中存在其他磁盘分区，如Windows系统分区或其他Linux分区，也可以通过“位置”菜单访问相应的文件系统。例如，在一个同时安装了Microsoft Windows与Ubuntu Linux双系统的计算机中，“位置”菜单中将会存在表示Windows系统分区的菜单项，其名字为磁盘分区的卷标，如“A22eXP”，单击磁盘分区菜单项即可安装、访问相应的文件系统，右击磁盘分区（或其他存储设备）后面的三角形图标，可以卸载相应的文件系统，如图2-35所示。



图2-35 磁盘分区界面

2.4.5 移动存储设备

GNOME桌面支持各种常用的移动存储设备，如软盘、CD/DVD及USB移动盘等。当插入CD/DVD或USB移动盘等存储介质时，系统会自动识别，同时把相应的文件系统安装到/media下面的某个子目录，然后在GNOME桌面中增加一个图标，在“位置”菜单中增加一个以文件系统卷标（或容量）命名的菜单项，最后弹出一个“文件浏览器”窗口，进入移动存储介质的根目录。此时可以浏览、读写、移动、复制或删除移动存储介质中的文件，实现数据的备份与交换等，如图2-36所示。



图2-36 访问移动存储介质

在Ubuntu Linux系统中，如果插入的是软盘，系统不会自动安装其中的文件系统，因而也不会GNOME桌面中增加软盘图标。此时可以选择“位置→计算机”菜单，在弹出的“计算机 - 文件浏览器”窗口中单击“软盘驱动器”图标，即可安装相应的文件系统，访问其中的文件。如果其他存储设备因故没有自动安装，也可采用同样的做法。

如果关闭了“文件浏览器”窗口，可以单击GNOME桌面中的相应图标，或选择“位置”菜单中的相应菜单项，打开文件浏览器，访问其中的文件。

在用完移动存储设备之后，可以右击相应的设备图标，或右击文件浏览器“位置”窗口中的相应存储设备，然后取决于所用存储设备的类型，从上下文菜单中选择“卸载”、“弹出”或“Safely Remove Drive”，取出移动存储介质。也可以直接右击文件浏览器“位置”窗口中存储设备后面的三角形图标，卸载相应的文件系统。



注意 移动存储介质的设备名因卷标的不同或因有无卷标而不同，安装的目录也会随之变化。

2.4.6 搜索文件

“搜索文件”菜单用于检索文件系统或存储设备中的文件，其效果等同于直接运行find命令。例如，如果想要查询系统中存在哪些配置文件，可以使用“*.conf”作为检索模式（大部分配置文件均以“.conf”作为文件扩展名），其结果如图2-37所示。



图2-37 “搜索文件”界面

2.5 系统菜单



图2-38 系统菜单

“系统”菜单主要提供系统的管理与维护功能，如网络管理、用户与用户组管理、打印管理、系统服务管理、系统日期与时间设置，软件源维护，以及更新软件包等。此外，还可以访问Ubuntu帮助中心，获取联机帮助与支持信息。访问GNOME网站，获取用户指南和系统管理员指南等文档。了解Ubuntu，了解其特点及由来等，如图2-38所示。

在“系统”菜单（或其他GNOME图形界面的程序）中，当选择的菜单项或功能只有超级用户才能执行时，许多按钮将会呈现暗灰色，此时可以单击钥匙状按钮（其右边附有“点击以进行变更”文字），在新弹出的“认证”窗口中输入sudo密码，然后才能继续。

2.5.1 首选项

作为系统管理工具，“首选项”主要用于设定各种GNOME桌面工具的默认参数或配置数据，如默认打印机，菜单的组织与布局，首选的应用程序，是否在菜单中显示图标，快捷键，显卡的分辨率与刷新率，系统空闲时是否启动屏幕保护程序，屏幕保护程序开始运行时是否锁住屏幕，以及桌面背景等。本节仅以四个例子予以说明。

1. 菜单布局

利用菜单布局功能，可以查询GNOME环境的菜单布局与配置信息，也可以根据自己的需要，定制菜单布局。例如，增加新的菜单，在现有的菜单中增加新的菜单项，隐藏甚至删除菜单项，上移或下移菜单项，调整菜单项的顺序，以及修改菜单项的属性等。

如果想要定制菜单布局，可以选择“系统→首选项→主菜单”，打开“主菜单”窗口，如图2-39所示。

例如，在“系统管理”菜单中，如果想把“打印”菜单项改为“打印管理”，可在“主菜单”界面左边的窗口中单击“系统管理”，从右边的窗口中选择“打印”，然后单击右键，在弹出的窗口中，把“名称”字段中的“打印”改为“打印管理”即可，如图2-40所示。



图2-39 “主菜单”窗口。



图2-40 修改菜单项属性

2. 启动应用程序

利用“系统→首选项→启动应用程序”菜单，可以设置用户注册时自动启动的应用程序，如网络管理器、音量控制以及电源管理等，如图2-41所示。

3. 快捷键

利用“系统→首选项→键盘快捷键”菜单，可以设置快捷键。实际上，GNOME已经提供了部分常用的快捷键，例如，使用“Print”键可以直接抓取整个屏幕的截图，使用“Alt-Print”组合键可以直接获取活动窗口的屏幕截图，而无需调用GMIP等截图工具，如图2-42所示。

4. 屏幕保护程序

“屏幕保护程序”用于设置、激活屏幕保护程序，以及在激活屏幕保护程序时是否锁定屏幕等，如图2-43所示。



图2-41 “启动应用程序首选项”界面



图2-42 GNOME快捷键设置



图2-43 “屏幕保护程序首选项”窗口

2.5.2 系统管理

“系统管理”主要用于管理、配置及维护系统，如设置打印机、系统日期与时间、用户与用户组、软件源及语言支持等，也可用于更新软件及查询系统日志等。

1. 打印管理

在系统会话过程中，用户可能需要打印文件。GNOME桌面环境利用通用UNIX打印系统（Common UNIX Printing System，CUPS），管理和维护可用的打印机，包括本地和网络打印机。

Ubuntu Linux系统支持众多流行品牌及型号的打印机，如作者本人的HP PSC 2310打印机。打印机可以通过串口、并口或USB接口连接的本地打印机，也可以是网络打印机，如IPP（Internet Printing Protocol）打印机或Samba服务器提供的打印机等。

连接USB接口的打印机相对比较简单。例如，把HP PSC 2310打印机连接到系统的USB接口之后，打开打印机电源，系统就会自动检索、识别（信息公告区将会出现一个打印机图标）以及配置打印机。最后通过一个消息框，通知用户打印机已经配置就绪。

如果连接的是串口或并口打印机，或采用网络打印机，通常需要自己配置打印机。此时可以选择“系统→系统管理→打印”菜单，进入“打印机配置”界面，如图2-44所示。

单击左上角的“新建”按钮，在弹出的“新打印机”界面中选择连接方式，如串口、并口或网络打印机等，如图2-45所示。

取决于连接方式，后面的设置步骤和方法有所不同，读者可以根据设置向导，选择打印机品牌和型号（用于安装驱动程序），设置打印机的名字（用于指定打印机），指定网络打印机的位置（如主机名和端口号等），以及设置打印机的访问控制等。

一旦配置好打印机，即可在文本编辑器等应用程序窗口中选择“文件→打印”菜单，或在命令行中使用lp等命令，利用选定的打印机打印文件。必要时也可以选择“系统→首选项→默认打印机”菜单，设置系统默认的打印机，如图2-46所示。



2. 更新管理器与新立得软件包管理器

“更新管理器”与“新立得软件包管理器”均用于维护软件，安装与更新软件包，详见第13章“软件管理”。

3. 软件源

利用“软件源”，可以查询、设置及维护Ubuntu Linux系统的软件源。Ubuntu提供上百个软件源，选用一个快速的软件源，能够提高软件下载及系统更新的速度，从而节省网络费用。如果不知道哪一个软件源更快，更适合自己的，可以在图2-47所示的软件源界面中单击“下载自”下拉框，选择“其他”菜单，然后单击“选择下载服务器”界面右边的“选择最佳服务器”按钮，由系统负责选择一个高速的软件源（有关软件源的说明及其使用，详见第13章“软件管理”）。

4. 时间和日期

“时间和日期”用于显示或设置系统时间，设置或启用网络时间协议NTP，连接NTP服务器，实现时间同步，以及显示或设置时区等，如图2-48所示。



图2-47 “软件源”界面



图2-48 “时间和日期设置”界面

5. 网络工具

“网络工具”用于执行常规的网络操作，如查询网络信息，测试网络连接状态，跟踪网络路由，扫描网络端口等，如图2-49所示。



图2-49 “网络工具”界面

6. 系统监视器

“系统监视器 (System Monitor)”类似于Microsoft Windows系统中的任务管理器，可用于查询系统的硬件配置；监控、改变进程的运行状态（如暂停、恢复运行或终止进程等），修改进程的优先级，查询进程的内存映像及其打开的文件等；监控系统资源的使用情况，其中包括CPU、内存与交换区的利用率，以及网络的流量等；查询已安装文件系统的使用情况，包括已经占用及空闲的存储空间等，如图2-50所示。

7. 系统日志

“系统日志”用于查询各种系统日志，包括位于/var/log目录中的主要日志文件messages中的信息，以及authlog、dmesg、kern.log和syslog等日志文件中的信息，如图2-51所示。

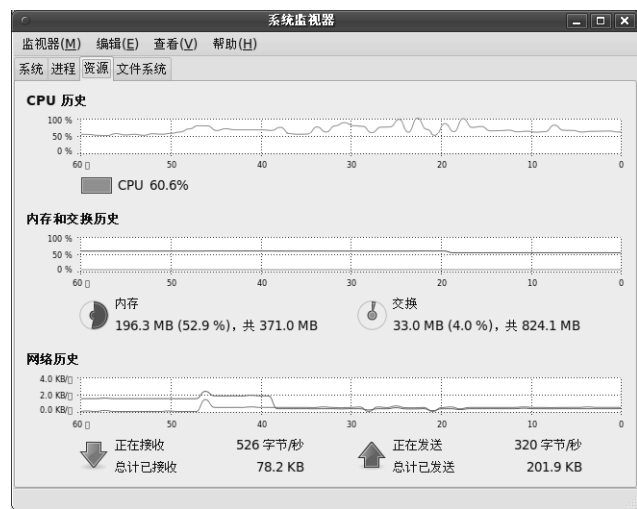


图2-50 “系统监视器” 界面

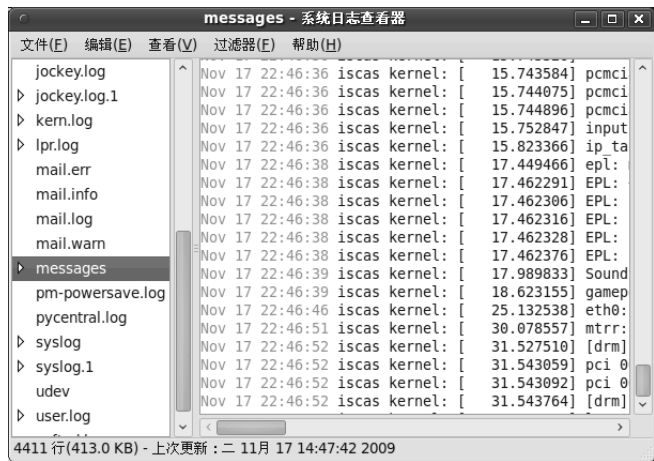


图2-51 “系统日志查看器” 界面

8. 用户和组

“用户和组”是一个用户管理工具。系统管理人员可以利用这个图形界面增加、删除或修改用户与用户组的有关信息。普通用户可以查询，但只有超级用户才能执行用户的增、删、改等操作。有关用户或用户组的管理，详见第9章“用户管理”。

9. 语言支持

“语言支持”用于查询系统当前支持的语言，设置菜单和窗口界面中使用的语言，设置注册界面中使用的语言环境，以及安装附加的语言支持软件包等，如图2-52所示。

10. 制作USB启动盘

“USB启动盘创建器”用于制作USB启动盘。制作时，可以利用刻录的CD/DVD安装介质或ISO映像文件作为源数据。在Linux系统不断升级换代的情况下，利用USB启动盘引导、安装Ubuntu Linux系统是非常方便的，如图2-53所示。



图2-52 “语言”界面



图2-53 “制作启动盘”界面

2.6 定制GNOME桌面环境

根据自己的爱好，用户可以定制GNOME桌面环境，如自己设定主题、桌面背景、面板的布局与位置等。

2.6.1 定制面板

GNOME桌面环境的整体布局与内容均可定制，菜单面板和窗口面板的位置以及其中的内容，都可以根据用户的需要进行调整。例如，为了调整菜单面板或窗口面板的位置，可以使用鼠标上、下、左、右拖动面板，把面板拖到桌面四周的任何位置，以竖条形式垂直显示面板，也可以在屏幕左右增加新的面板，增加菜单、图标与应用程序启动器等。当选用多个面板时，可以按功能划分面板，分类存放菜单、图标以及应用程序启动器等。至于究竟如何定制面板，可以依据个人的品味与爱好而定。

如果想要增加一个面板，可以在现有面板的任何空白处右击，从弹出的上下文菜单中选择“新建面板”菜单项。若想增加图标与应用程序启动器等，可以从弹出的上下文菜单中选择“添加到面板”菜单项，然后从“添加到面板”对话框中选择需要增加的项目。

2.6.2 定制桌面背景

通过修改主题、背景或界面等方式，也可以定制桌面区，使之包含需要频繁执行的程序或任务；也可以为桌面定制独特的外观界面，使得每次注册到系统时，一眼就能够看出自己的工作环境。此外，也可以随时修改桌面，以满足用户的需求。

如果想要更换GNOME桌面的背景图像，可以在桌面区的空白位置右击，从桌面区的上下文菜单中选择“更改桌面背景”菜单项，从弹出的“外观首选项”窗口中选择一个新的壁纸作为GNOME桌面背景，选中后将会立即生效，最后单击“关闭”按钮结束。

另外一种方法是从GNOME桌面中选择“系统→首选项→外观”菜单，在“外观首选项”窗口中选择“背景”标签，其他步骤如前所述，如图2-54所示。



图2-54 定制桌面背景

2.6.3 定制菜单面板

为了快速启动常用的应用程序和工具软件，可以把相应的图标加到菜单面板中。在GNOME桌面环境中，能够加到菜单面板的图标分为两种：一种是用于快速启动应用程序的按钮；第二种是能够实时显示各种状态信息的Applet。

Applet是一种小型程序，能够以图像或文字形式实时显示信息。GNOME随机提供许多内置的Applet，如“CPU频率范围监视器”、“气象报告”、“时钟”及“系统监视器”等。也可以从网站下载Linux程序员开发的，且适用于GNOME的Applet。

按钮是用于访问常用操作和功能的特殊图标。GNOME提供的按钮包括“搜索文件”、“注销”、“关机”、“字典查阅”及“运行应用程序”等。

其中，“搜索文件”按钮用于启动文件检索工具，在系统中检索匹配指定模式的文件或文件夹，检索包含指定文字内容的文件，或按一定属性（如修改日期、文件大小或文件属主等）检索文件等。

如果想把系统内置的图标（包括Applet与按钮）加到菜单面板中，可以右击菜单面板的空白区域，从弹出的上下文菜单中选择“添加到面板”菜单项，然后从弹出的“添加到面板”窗口中选择期望的功能，如“眼睛”和“气象报告”，单击“添加”按钮后，再单击“关闭”按钮，如图2-55所示。

“气象报告”图标通常能够显示世界各地当天的即时天气状况，单击该图标时还可以显示详细的气象信息，如相对湿度、气压、能见度、日出和日落时间等。注意，增加图标后需要右击图标，在上下文菜单



图2-55 在菜单面板中加图标

中选择“首选项”，从弹出的“天气首选项”窗口中选择“位置”标签，然后选择所在的城市（小城市暂时不一定能够提供数据）。增加“眼睛”与“气象报告”图标后的效果如图2-56所示。

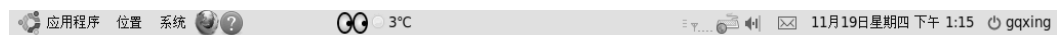


图2-56 菜单面板中的“眼睛”与“气象报告”图标

第3章 命令行基础知识

本章采用Linux系统默认的命令解释程序Bash，从最基本的命令行界面入手，介绍Shell的命令行结构、标准输入与输出、输入输出重定向、管道、命令历史与命令别名，以及作业控制等。

Linux系统提供大量的命令和工具，如能熟练地掌握最基本的命令，灵活地利用系统提供的各种机制，组合运用Linux系统的命令和工具，就能够充分地发挥Linux系统的潜能。Linux系统的强大功能完全体现在其命令行环境中，图形界面（如GNOME桌面系统）提供的所有功能实际上也是利用基本的命令和工具实现的。因此，熟练地掌握、灵活自如地运用一定数量的常用命令和工具是每个学习Linux系统的人都应当具有的基本功。

若想进入命令行环境，在注册到系统之后，可以从“应用程序”菜单中选择“附件→终端”，即可进入终端仿真窗口，利用命令行界面访问Linux系统。此外，在终端窗口的命令行操作过程中，可以随时使用鼠标选中屏幕上的文本数据，利用右键的上下文菜单，复制、剪切和粘贴选中的文本数据，如图3-1所示。



图3-1 终端窗口。

从本章开始，初学者最好使用安装Ubuntu Linux系统时创建的用户账号，以普通用户的身份注册到Linux系统中，在自己的主目录中开始学习Linux系统的各种命令和工具，尽量不要动用系统文件，等到有了一定的基础时，再以超级用户root的身份管理Linux系统，以免无意中损坏系统。

3.1 命令行结构

在Linux系统中，一个命令通常由命令名、命令选项和命令参数等三部分内容组成，中间以空格或制表符等空白字符隔开。命令形式如下：

<命令名> <命令选项> <命令参数>

其中，命令选项通常是以减号“-”开始的单个字符。与UNIX系统不同的是，Linux系统还提供以双减号“--”为起始标志的命令选项（通称GNU选项），其选项通常是可以按字面意义

理解的单个英文单词，或是由单词加连字符组成的词组。除了个别命令选项（如“-help”选项）之外，以双减号“--”为起始标志的命令选项大多是单字符命令选项的同义词，具有相同的意义，因而可以替换使用。

顾名思义，命令选项是可以省略的。同样，命令参数也可以省略。也就是说，在命令行结构中，只有命令名通常是必须提供的。一个最简单的命令可以仅仅包含命令名本身。在这种情况下，命令选项和参数均采用默认值。下面就是一个最简单的命令，其中的命令选项和参数均采用默认值，即列出系统当前的日期和时间。

```
$ date
2009年 11月 15日 星期日 10:31:12 CST
$
```

在实际应用过程中，可以根据具体要求，视情况选用或省略命令选项和命令参数。而且，命令选项和命令参数能够与命令名灵活地组合使用。例如，下列命令仅由命令名和一个命令选项“-n”组成，省略了命令参数，其作用是列出系统的名字。

```
$ uname -n
iscas
$
```

下列命令由命令名和命令参数组成，而省略了命令选项，其作用是以简单的输出形式列出指定目录下的文件。

```
$ ls /etc/network
if-down.d if-post-down.d if-pre-up.d if-up.d interfaces
$
```

取决于命令本身，命令参数可以是目录、文件或其他内容。例如，在下列命令中，ls是命令的名字，“-l”是命令的选项，/etc/profile文件是命令的参数：

```
$ ls -l /etc/profile
-rw-r--r-- 1 root root 497 2009-10-29 05:49 /etc/profile
$
```

下列命令使用了“-l”和“-a”两个命令选项，但省略了命令参数。其作用是列出当前目录中的所有文件，包括隐藏文件。

```
$ ls -la
总计 7866
drwxr-xr-x  40 qqxing qqxing    4096 2009-11-15 16:18 .
drwxr-xr-x   4 root  root    4096 2009-11-15 16:12 ..
-rw-----   1 qqxing qqxing    248 2009-11-15 15:56 .bash_history
-rw-r--r--   1 qqxing qqxing    220 2009-11-15 06:43 .bash_logout
-rw-r--r--   1 qqxing qqxing   3180 2009-11-15 06:43 .bashrc
.....
$
```

命令选项主要用于限定命令的具体功能，同时也决定了命令的最终运行结果。在Linux系统中，每个命令通常均提供大量的选项，因而具有丰富的功能。选项可以单独给出，也可以组合使用。如果选项本身也带有参数，则这样的选项及其参数必须单独列出。在下列排序命令中，

因为“-k”和“-o”等命令选项本身也要求提供参数，故需要分别给出：

```
$ sort -k 5 -n -o sorted tobesorted
$
```

其中，“-k 5”中的5就是命令选项“-k”的参数，表示以第5个字段为关键字进行排序。“-n”选项表示按数值的大小排序。“-o sorted”中的sorted也是选项“-o”的参数，表示存储最终排序结果的输出文件。最后的tobesorted则是命令参数，表示需要排序的输入文件。

在后续章节介绍的Linux命令语法格式中，凡以方括号“[]”形式给出的命令选项均为可选项，因而可视具体需求选择使用或忽略。

在Linux系统的命令提示符下，一次通常仅输入一个命令。如果愿意，也可以一次输入多个命令，命令之间用分号隔开。例如，下列两个命令的作用是首先进入指定的目录/etc/network，然后列出其中的文件。

```
$ cd /etc/network; ls -l
总计 20
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 if-down.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:59 if-post-down.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:59 if-pre-up.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:57 if-up.d
-rw-r--r-- 1 root root 134 2009-11-15 14:58 interfaces
$
```

另外，也可以使用圆括号把若干命令合并在一起，使之构成一个组合命令。

```
$ (cd /etc/network; ls -l)
总计 20
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 if-down.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:59 if-post-down.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:59 if-pre-up.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:57 if-up.d
-rw-r--r-- 1 root root 134 2009-11-15 14:58 interfaces
$
```

除了括号之外，上述两种命令形式完全一样，有时其效果也完全一样。但两者的意义却大不相同。第一种命令形式只是在一个逻辑行上并列输入了多个命令，其效果同一次输入一个命令基本上没有区别，而且都是在当前Shell下运行。而第二种命令形式则把多个命令看做一个组合命令，在一个子Shell中运行，所有命令的输出数据将会合并为一个输出流，其差别在管道操作中尤为明显。

例如，下列两组命令的最终结果是完全不同的（其中，“wc -l”命令用于计算读入的行数）。首先让我们分别观察date与who两个命令的输出。

```
$ date
2009年 11月 15日 星期日 10:36:12 CST
$ who
gqxing tty7 2009-11-15 15:05 (:0)
gqxing pts/0 2009-11-15 16:23 (:0.0)
gqxing pts/1 2009-11-15 16:26 (169.254.78.56)
$
```

然后观察两个并列命令的输出结果:

```
$ date; who
2009年 11月 15日 星期日 10:36:18 CST
gqxing  tty7      2009-11-15 15:05 (:0)
gqxing  pts/0      2009-11-15 16:23 (:0.0)
gqxing  pts/1      2009-11-15 16:26 (169.254.78.56)
$
```

接着，再使用管道把两个并列的命令与计算输入数据行数的wc命令连接起来，观察其输出结果。可以看到，wc命令仅仅计数了who命令的输出结果——3行。

```
$ date; who | wc -l
2009年 11月 15日 星期日 10:36:25 CST
3
$
```

最后，再利用圆括号把两个命令组合到一起，通过管道连接wc命令，观察其结果。可以看到，两个命令各自的输出数据已合并到一起，wc命令计数的最终结果是4行。

```
$ (date; who) | wc -l
4
$
```

如果命令较长，超出一个物理行的宽度，可以使用反斜线“\”把命令写到多个物理行上。也可以继续输入，由系统自动延伸至后续行上。例如，下列read命令提示用户依次输入名字、电话号码和电子邮件地址，然后分别存于name、phone和email三个变量中。由于命令行较长，其中采用了反斜线“\”，把超长部分延续到下一行：

```
$ read -p "Please input name, phone and email address in order: " \
name phone email
$
```

如前所述，许多GNU实用程序都支持以双减号“--”为起始标志的选项。这些选项或者是原有单减号“-”选项的另外一种表现形式，或者是命令功能的扩充。例如，sort命令的“-k”选项对应的双减号选项为“--key”。如果使用GNU命令形式，则可以把前述的sort命令改写如下：

```
$ sort --key=5 -n -o sorted tobesorted
$
```

3.2 后台进程

在Linux系统中，Shell通常以前台形式解释执行用户输入的命令。在Shell的命令提示符（超级用户的默认命令提示符为“#”，普通用户的默认命令提示符为“\$”）下，系统将会等待用户输入命令，直至用户按下Enter键。然后由Shell解释命令行，创建一个新的进程，执行用户提交的命令，最后给出命令的执行结果。

在Shell解释执行命令期间，用户需要等待命令执行完成，中间不能做任何事情。即使一个命令的执行时间过长，中间不需要输入任何信息，不需要监控命令的执行过程时，也只能静等命令执行的完成。

为了解决此问题，Shell提供了后台进程机制，允许用户以后台进程的方式运行命令，而无需等待命令执行的完成。在解释命令行，以后台进程方式执行命令的同时，Shell将会立即输出命令提示符，等待用户输入新的命令，从而并发地运行多个命令。

为了以后台进程方式运行命令，只需在命令的后面增加一个“&”符号即可。例如，为了在系统中检索究竟存在多少个名为core的文件，可以使用下列形式的命令：

```
$ find /home -name core -print &
[1] 2886
$
```

上述命令行后面的“&”符号告诉Shell，提交的命令应以后台进程的方式执行，无需等待命令执行的完成。因此，在给出作业号和进程ID之后，系统将会立即输出命令提示符，提示用户进一步输入其他命令。

在上述例子中，方括号中的“1”是以后台作业方式运行的find进程的作业号，“2886”是find进程的PID。为了跟踪与控制后台作业，可以利用作业控制命令控制作业的运行状态。例如，可以使用作业控制命令fg把后台作业转为前台进程继续运行。有关后台作业控制的详细介绍，参见本章“3.10 作业控制”一节。也可以使用进程控制命令（如kill等命令），利用进程ID，或根据ps命令获取进程信息控制进程的运行行为。有关进程控制的介绍，详见第10章“进程管理”。

如果后台进程有输出数据，其输出信息将会随时出现在用户的终端屏幕上。注意，后台进程的输出信息有可能会出现在交互会话期间的任何时刻，造成屏幕输出的混乱。例如，如果用户正在使用vi/vim编辑文件，后台进程的输出很可能会干扰编辑器的正常工作。

3.3 标准输入、输出与错误输出

在Linux系统中，任何命令，包括Shell本身，通常总是读取来自终端键盘输入的数据，这个数据输入源称做标准输入（stdin），其文件描述符为0。命令的运行结果通常总是输出到用户终端的屏幕上，这个输出目的称做标准输出（stdout），其文件描述符为1。另外，在命令的执行期间，如果出现问题，相应的错误信息也将输出到用户终端的屏幕上，这个输出目的通常称做标准错误输出（stderr），其文件描述符为2。

一旦注册到系统中，系统总是为用户打开三个默认的文件：标准输入（键盘）、标准输出（终端屏幕）和标准错误输出（也是终端屏幕，用于输出错误信息），如图3-2所示。

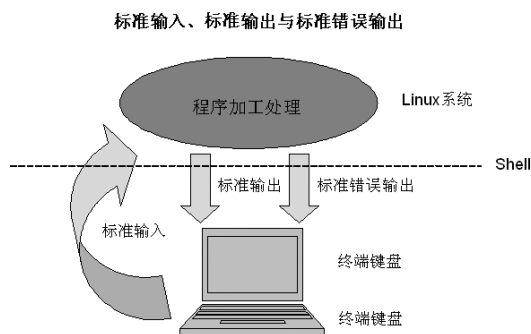


图3-2 标准输入、标准输出和标准错误输出与终端键盘和屏幕间的关系

例如，当用户从键盘上输入下列uname命令时，其输出信息将出现在用户的终端屏幕上。

```
$ uname -a
Linux iscas 2.6.31-14-generic #48-Ubuntu SMP Fri Oct 16 14:04:26 UTC 2009 i686 GNU/Linux
$ uname -n
iscas
$ uname -s
Linux
$
```

3.4 I/O重定向

为了仔细分析命令的处理结果，有时需要把命令的标准输出保存到某个文件中。这就需要用到Shell的输入输出重定向机制。例如，利用输出重定向符号“>”，下列命令行可以把“ls -l”命令的输出结果保存到指定的文件中：

```
$ ls -l > fname
$
```

在上述命令中，“>fname”意味着把“ls -l”命令的输出数据重定向并写到指定的文件fname中。如果指定的文件不存在，Shell将会创建一个新的文件，然后把输出数据保存到其中。如果文件已经存在，文件中原有的内容将会被清除，并代之以命令的输出数据。

为了保留文件中原有的数据，把命令的输出数据附加到文件的后面，可以使用重定向符号“>>”：

```
$ ls -l >> fname
$
```

其中，“>>fname”意味着把“ls -l”命令的输出数据重定向并附加到指定文件fname的后面。同样，如果指定的文件不存在，Shell将会创建一个新的文件，并把输出数据保存到其中。如果文件已经存在，Shell将会把命令的输出数据附加到文件的后面，以保留文件中原有的内容。

任何命令（包括Shell本身）的标准输入也可以重定向，使命令直接读取某个文件而不是键盘输入。例如，wc命令的功能是读取标准输入中的输入数据，分别计数输入数据中的字符数、字数和行数。为了使wc命令能够直接读取某个文件中的数据内容，可以使用重定向符号“<”，使之直接读取指定的文件：

```
$ wc -l < fname
42
$
```

在Linux系统中，标准输入、标准输出和标准错误输出三个文件通常总是打开的。这三个文件，包括其他打开的文件均可做I/O重定向处理。所谓的I/O重定向，只不过是由Shell读取或捕捉来自文件、命令、程序或脚本中的输出，作为输入或输出信息传递给另外一个文件、命令、程序或脚本。

在Linux系统中，系统将会为每一个打开的文件分配一个文件描述符，每个文件都与一个文件描述符相关联。文件描述符是一个数字，便于Linux系统跟踪打开的文件（可以把文件描述符看做简化的文件指针）。标准输入、标准输出和标准错误输出的文件描述符分别是0、1和2。

作为一种临时性的双向机制，把其他打开文件的描述符分配到标准输入、标准输出和标准错误输出，然后利用这些文件描述符执行输入输出，有时是非常有用的（注意，使用文件描述符5可能会引起问题，这是因为当Shell利用exec命令创建子进程时，子进程将会继承文件描述符5。因此，最好避开这个特殊的文件描述符）。

表3-1给出了Shell支持的各种I/O重定向的规定及其说明。

表3-1 Shell I/O重定向

I/O重定向	简单说明
<i><fname</i>	使用指定的文件作为标准输入（其文件描述符为0），以便从指定的文件中接收输入数据
<i>>fname</i>	使用指定的文件作为标准输出（其文件描述符为1）。如果文件不存在，则创建命名的文件。如果文件存在，且noclobber标志已经设置，将会产生错误；否则，将会清除文件中原有的数据内容。参见第7章“Shell基础知识”中介绍的set命令
<i>> fname</i>	除了忽略noclobber标志之外，其功能与“>fname”相同
<i>>>fname</i>	使用指定的文件作为标准输出。如果文件存在，则把输出内容附加到文件后面；否则，创建指定的文件
<i><>fname</i>	以读写方式打开指定的文件，并使之作为标准输入
<i><<[-]fstr</i>	以指定的标志字符串fstr之后的文档（称做Here文档）作为标准输入，从fstr之后逐行读取数据，直至遇到第二个fstr（或EOF）标志。此时，第一个fstr是标准输入的起始标志，第二个fstr（或EOF）是标准输入的结束标志。如果“<<”后面附带减号“-”标志字符，则Shell将会忽略后随文本行前面的制表符
<i><&digit</i>	使用指定的文件描述符复制一个标准输入
<i>>&digit</i>	使用指定的文件描述符复制一个标准输出
<i><&-</i>	关闭标准输入。而“n<&-”则表示关闭输入文件描述符n
<i>>&-</i>	关闭标准输出。而“n>&-”则表示关闭输出文件描述符n
<i><&j</i>	把标准输入重定向到文件描述符j表示的输入文件中
<i>>&j</i>	把标准输出重定向到文件描述符j表示的输出文件中
<i>&>name</i>	把标准输出和标准错误输出均重定向到指定的文件中

如果I/O重定向符号“<”或“>”前面有一个数字，则表示相应的文件描述符（默认值分别为0或1）对应的文件，如表3-2所示。

表3-2 Shell I/O重定向

I/O重定向	简单说明
0 <i><fname</i>	把标准输入重定向到指定的文件中
1 <i>>fname</i>	把标准输出重定向到指定的文件中
1 <i>>>fname</i>	把标准输出重定向并附加到指定的文件中
2 <i>>fname</i>	把标准错误输出重定向到指定的文件中
2 <i>>>fname</i>	把标准错误输出重定向并附加到指定的文件中

I/O重定向	简单说明
<code>i>j</code>	把文件描述符 <i>i</i> 表示的输出文件重定向到文件描述符 <i>j</i> 表示的文件
<code>[j]<fname</code>	以读写方式打开指定的文件，并把文件描述符分配到指定的文件。如果文件不存在，则创建该文件。如果未指定文件描述符 <i>j</i> ，则表示默认的文件描述符0，即标准输入

采用下列命令形式，可以把标准错误输出重定向到标准输出，使命令的错误信息和命令的实际输出数据均写到标准输出中，即在终端屏幕上显示：

```
command 2>&1
```

在下面的例子中，前三个echo命令的标准输出已重定向到script.log文件，因而其输出信息将会写到文件中，而不是出现在终端屏幕上。

```
$ LOGFILE=script.log
$ echo "This line is written to log file." > $LOGFILE
$ echo "This line is appended to log file." >> $LOGFILE
$ echo "This line is appended to log file also." >> $LOGFILE
$ echo "This line is echoed to stdout."
This line is echoed to stdout.
$ cat script.log
This line is written to log file.
This line is appended to log file.
This line is appended to log file also.
$
```

在下面的例子中，前两个命令（变量赋值语句除外）的标准错误输出已重定向到script.errors文件，因此，命令的错误信息将会记录到指定的文件而不是写到终端屏幕上。而第三个命令的错误信息将会写到标准错误输出，即终端屏幕上，而不会出现在script.errors文件中。

```
$ ERRFILE=script.errors
$ bad_command1 2>$ERRFILE
$ bad_command2 2>>$ERRFILE
$ bad_command3
```

采用下列形式，可以把多个I/O重定向组合到一个命令行中。

```
command <input-file >output-file
```

采用下列形式，可以把标准输出和标准错误输出重定向到同一个文件中。

```
command >command.log 2>&1
```

其中，命令的任何错误信息将会写到文件command.log文件中，因为标准错误输出已经重定向到该文件。



I/O重定向的顺序是非常重要的。Shell将会根据文件描述符（文件）出现的顺序决定I/O重定向的关联关系。例如，下列I/O重定向命令意味着把命令的标准输出和标准错误输出均重定向到给定的文件中：

```
command 1>fname 2>&1
```


假定I/O重定向之前命令的标准输入、标准输出和标准错误输出对应的文件描述符如图3-3所示。

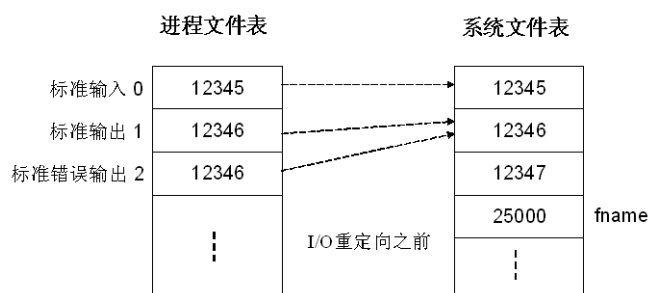


图3-3 I/O重定向之前

Shell将首先建立文件描述符1（即终端屏幕）与文件fname之间的重定向关系，也即把fname对应的文件描述符25000写到用户打开文件表中的标准输出位置。然后建立文件描述符2与文件描述符1表示的文件（即fname）之间的重定向关系，也即把标准输出对应的文件描述符25000写到用户打开文件表中的标准错误输出位置。最终的结果是，命令的标准输出和标准错误输出内容均写到文件fname中，如图3-4所示。

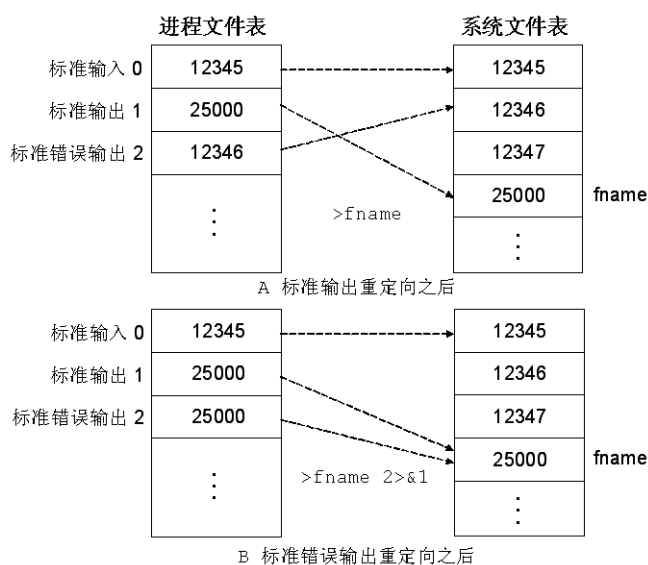


图3-4 I/O重定向之后（1）

如下所示，如果I/O重定向的顺序相反，则最终结果将会是另外一个样子。

```
command 2>&1 1>fname
```

假定I/O重定向之前命令的标准输入、标准输出和标准错误输出对应的文件描述符仍如图3-2所示。文件描述符2首先与文件描述符1建立重定向关系。由于两者对应的文件描述符相同（均为12346），故标准错误输出对应的文件描述符保持不变。然后，文件描述符1再与给定的文件fname建立重定向关系，把fname对应的文件描述符25000写到命令打开文件表中的标准输出位置。

最终结果是，命令的标准错误输出将会写到标准输出，即在终端屏幕上显示。而命令的标准输出则会写到给定的文件fname中，如图3-5所示。

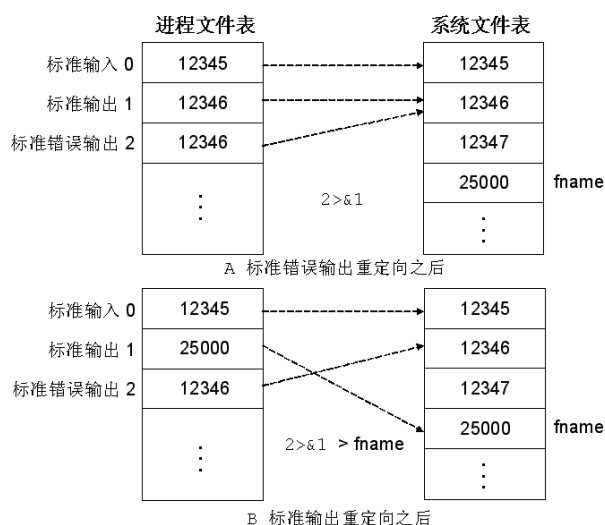


图3-5 I/O重定向之后（2）

由此可见，如果I/O重定向的顺序不同，最终的结果将会完全不一样。例如，执行下列命令时，错误信息将会输出到标准输出，即终端屏幕上，而不会写到文件command.log中。

```
$ ls -yz 2>&1 >> command.log
ls: invalid option - 'y'
Try 'ls --help' for more information.
$
```

利用下列命令和I/O重定向的方法，可以创建一个新的空文件。如果文件存在，则清空文件fname中原有的内容。

```
$ > fname
$
```

利用I/O重定向的方法，还可以创建一个新文件，使之包含当前目录下的目录文件树列表。例如：

```
$ ls -lR > dir-tree
$
```



子进程能够继承已经打开的文件描述符。为了防止文件被子进程继承，应注意随时关闭不再需要的文件描述符。

3.5 管道

在Linux系统中，管道是一种先进先出的单向数据通路。利用管道符号“|”，可以把一个命令的标准输出连接到另一个命令的标准输入。例如，利用管道把ls和wc两个命令连接到一起，可以获知指定目录下的文件数量（“-w”选项表示以字为单位进行计数）：

```
$ ls /usr | wc -w
9
$
```

从上述命令的最终执行效果看，可以把组合命令分解为以下两个命令：

```
$ ls /usr > fname; wc -w < fname
9
$
```

由此可以看出，当使用管道方式连接两个命令时，Shell将会把两个进程连接起来，利用管道的单向通信特征，把一个进程的标准输出传递到另一个进程的标准输入。Shell将会协调两个进程的同步，使两个进程能够并发地运行，这样就可以省略存储中间处理结果的临时文件。实际上，管道是一种特殊的I/O重定向。

管道的常见用法是为滤通程序提供原始数据，由滤通程序读取来自标准输入的数据，按照指定的检索原则和模式，从输入数据中提取期望的、包含给定字符串的数据。在Linux系统中，grep就是这样一个常见的滤通程序。

例如，为了从ps命令输出的众多进程中找出某个特定的进程，可以使用管道连接ps和grep命令：

```
$ ps -ef | grep cron
root      943      1  0 Nov16 ?          00:00:00 cron
gqxing    3150    3126  0  01:29 pts/3      00:00:00 grep --color=auto cron
$
```

另外一个常见的用法是利用管道把进程的输出数据传递给sort命令，使之按照一定的排序原则进行排序，最终输出排序后的结果。例如，下列组合命令最终将会按照字符顺序输出注册的用户：

```
$ who | sort
cathy     pts/2      2009-11-17 01:32 (169.254.78.56)
gqxing    pts/0      2009-11-17 01:33 (:0.0)
gqxing    pts/1      2009-11-17 01:34 (:0.0)
gqxing    tty7       2009-11-16 22:53 (:0)
$
```

利用管道，可以把多个命令组合到一起，把命令的标准输出依次传递到下一个命令的标准输入，最终得到经过多个命令依次处理的结果：

```
command1 | command2 | command3 > output-file
```

若想依次加工处理多个命令、脚本和程序的输出数据，管道是非常有用的。例如，为了获取cron的进程ID，以便使用kill命令终止该进程，可以利用管道连接下列命令，首先从进程列表中提取与cron有关的进程，然后删除可能存在的“grep cron”进程，最后再使用gawk命令截取位于第2个字段的进程ID（参见第10章“进程管理”介绍的简化版的pgrep和pidof命令）：

```
$ ps -ef | grep cron | grep -v grep | gawk '{print $2}'
843
$
```

为了复制和备份一个完整的目录，也可以利用管道，组合使用find和cpio两个命令，把当前目录下的所有目录和文件按照原有的目录层次结构复制到一个新的目录位置。

```
$ cd sourcedir
$ find . -print | cpio -pduv newdir
.....
$
```

Linux系统还提供一个相当于三通管的实用程序tee。tee命令的主要功能是通过标准输入接收并显示数据，同时把数据存储在指定的文件中。因此，可以利用tee命令，在显示一个命令输出数据的同时，把输出结果存储到一个指定的文件中，以便将来再检查。例如，为了显示并保存当前所有注册用户的列表文件，可以使用下列命令（其效果如图3-6所示）：

```
$ who | tee userlist
gqxing  tty7      2009-11-18 19:31 (:0)
gqxing  pts/0      2009-11-18 19:44 (:0.0)
gqxing  pts/1      2009-11-18 19:47 (169.254.78.56)
cathy   pts/2      2009-11-18 19:48 (169.254.78.56)
$
```

tee命令的“T”型三通功能

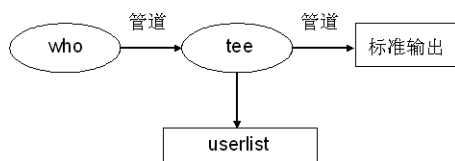


图3-6 tee命令功能图解

3.6 元字符与文件名生成

在Linux系统中，很多命令均使用文件作为命令参数。例如，下面的ls命令用于列出指定文件参数atmmon.c的访问权限、文件大小及文件属主等有关属性：

```
$ ls -l atmmon.c
-rw-r--r-- 1 gqxing gqxing 60978 2009-11-17 18:03 atmmon.c
$
```

当需要处理一组具有共同属性的文件时，怎样指定文件名参数呢？Shell提供一种文件名生成机制，使用户能够利用元字符（或称通配符）实行模式匹配，最终生成一个具有同一属性的文件列表。

为了简化手工输入，按照某种模式选择具有同一属性的文件，Shell的文件名生成机制是非常有用的。例如，可以使用下列命令列出当前目录中的所有C程序文件：

```
$ ls -l *.c
-rw-r--r-- 1 gqxing gqxing 30235 2009-11-17 17:56 atmcom.c
-rw-r--r-- 1 gqxing gqxing 60978 2009-11-17 18:03 atmmon.c
-rw-r--r-- 1 gqxing gqxing 82468 2009-11-17 18:04 handler.c
-rw-r--r-- 1 gqxing gqxing 58257 2009-11-17 18:02 listener.c
$
```

Shell命令使用生成的文件名作为参数，依次处理每一个文件。上述命令的处理结果就是依次列出当前目录下每一个以“.c”为文件名后缀的C程序文件。其中的星号“*”就是一个元字符，可以匹配任何字符或字符串，包括空字符串。表3-3给出了Shell支持的与文件名生成有关的元字符及其说明。注意，元字符可以组合使用。

表3-3 与文件名生成有关的元字符

元字符	简单说明
*	可以匹配任何数量的字符或字符串，包括空字符串。例如，“ <code>boc*</code> ”表示任何一个以“ <code>boc</code> ”为起始字符的字符串，“ <code>*.c</code> ”表示任何一个以“.c”为文件名后缀的C程序文件
?	可以匹配相应位置的任何一个字符。例如，“ <code>file?</code> ”表示任何一个以“ <code>file</code> ”为起始字符，后面附加单个字符的字符串
[...]	由方括号定义的字符集或字符范围，可以使用其中任何一个字符匹配文件名相应位置的一个字符。方括号中的字符集可以由任何字符组成，数量不限。字符可以一一列举，也可以在两个字符之间加一个减号“-”，表示一个字符范围。例如， <code>[a-z]</code> 表示所有的小写字母， <code>[0-9]</code> 表示0至9之间的任何数字
[!...]或[^...]	如果方括号中的第一个字符是感叹号“!”或上箭头“^”，则其意义恰好相反，表示可以匹配任何一个不属于给定字符集范围的字符

假定当前目录存在下列文件，现在通过例子，进一步说明怎样使用元字符匹配文件名，解释Shell的文件名生成机制。

```
$ ls -l
总计 28
-rwxr-xr-x 1 gqxing gqxing 255 2009-11-18 13:36 Chkfs
-rwxr-xr-x 1 gqxing gqxing 379 2009-11-18 13:36 chkroot
-rwxr-xr-x 1 gqxing gqxing 941 2009-11-18 13:31 ftpget
-rwxr-xr-x 1 gqxing gqxing 216 2009-11-18 13:30 menu1
-rwxr-xr-x 1 gqxing gqxing 125 2009-11-18 13:30 menu2
-rwxr-xr-x 1 gqxing gqxing 802 2009-11-18 13:29 settcp
-rwxr-xr-x 1 gqxing gqxing 333 2009-11-18 13:31 upload
$
```

按照表3-3的说明，`[a-z]`可以匹配任何小写字母，“*”可以匹配任何字符串，故下面的命令可以列出当前目录下任何以小写字母为起始字符的文件名。

```
$ ls -l [a-z]*
-rwxr-xr-x 1 gqxing gqxing 379 2009-11-18 13:36 chkroot
-rwxr-xr-x 1 gqxing gqxing 941 2009-11-18 13:31 ftpget
-rwxr-xr-x 1 gqxing gqxing 216 2009-11-18 13:30 menu1
-rwxr-xr-x 1 gqxing gqxing 125 2009-11-18 13:30 menu2
-rwxr-xr-x 1 gqxing gqxing 802 2009-11-18 13:29 settcp
-rwxr-xr-x 1 gqxing gqxing 333 2009-11-18 13:31 upload
$
```

由于问号“?”可以匹配任何一个字符，故下列`ls`命令可以列出当前目录中文件名前四个字符为`menu`，第五个字符为任何字符的所有文件:

```
$ ls -l menu?
```

```
-rwxr-xr-x 1 gqxing gqxing 216 2009-11-18 13:30 menu1
-rwxr-xr-x 1 gqxing gqxing 125 2009-11-18 13:30 menu2
$
```

为了列出当前目录中以s或u为首字符的所有文件，可以使用下列ls命令：

```
$ ls -l [su]*
-rwxr-xr-x 1 gqxing gqxing 802 2009-11-18 13:29 settcp
-rwxr-xr-x 1 gqxing gqxing 333 2009-11-18 13:31 upload
$
```

上述命令也可以改写如下：

```
$ ls -l s* u*
-rwxr-xr-x 1 gqxing gqxing 802 2009-11-18 13:29 settcp
-rwxr-xr-x 1 gqxing gqxing 333 2009-11-18 13:31 upload
$
```

为了列出当前目录中首字符为大写字母（或其他非小写字母）的所有文件，则可以使用下列三种ls命令形式之一：

```
$ ls -l [A-Z]*
-rwxr-xr-x 1 gqxing gqxing 255 2009-11-18 13:36 Chkfs
$ ls -l [!a-z]*
-rwxr-xr-x 1 gqxing gqxing 255 2009-11-18 13:36 Chkfs
$ ls -l [^a-z]*
-rwxr-xr-x 1 gqxing gqxing 255 2009-11-18 13:36 Chkfs
$
```

如上所述，问号“?”可以匹配当前目录中以单个字符命名的所有文件名。星号“*”能够匹配当前目录中的所有文件名（隐藏文件除外）。如果不存在能够匹配检索模式的文件名，指定的检索模式将会不加修改地作为参数传递给相应的命令。例如（仅当目录为空时，下面第二个例子才会出现如此的输出结果）：

```
$ ls -l ?
ls: cannot access ?: No such file or directory
$ ls -l *
ls: cannot access *: No such file or directory
$
```

元字符也可用于检索文件。例如，可以使用下列命令检索并列出/home/gqxing目录下任何子目录中名为core的文件。

```
$ echo /home/gqxing/*/core
```



任何元字符都不能匹配以句点“.”为首字符的隐藏文件名。换言之，以句点“.”为起始字符的隐藏文件名必须采用明显的匹配形式。例如，下列命令只能列出当前目录中的所有非隐藏文件：

```
$ echo *
Chkfs chkroot ftpget menu1 menu2 settcp upload
$
```

为了匹配以句点“.”为起始字符的隐藏文件名，可以采用下列命令形式，列出所有的隐藏文件：

```
$ echo .*
. . . .aptitude .bash_history .bash_logout .bashrc .....
$
```



set命令的“-f”选项（即“set -f”命令）能够禁止文件名的生成。当Shell无法解释元字符时，应注意检查是否设置了这个标志。

3.7 转义与引用

转义和引用是两个截然相反的概念。在Shell中，为了处理具有特殊意义的元字符，如“<”、“>”、“*”、“?”、“|”和“&”等，使之作为普通字符显示，可以采用转义符号“\”、单引号和双引号引用元字符，而引用的元字符则失去其特殊意义。

根据上述说明，本身具有特殊意义的元字符，如果在前面加上转义符号“\”，则失去其特殊意义。而对于某些普通字符，如果前面加上转义符号“\”，则具有特殊的意义，这些字符称做转义字符。表3-4给出了Shell支持的，具有特殊意义的部分转义字符，可用于echo等命令中，以便控制输出数据的显示格式等。

表3-4 Shell支持的部分转义字符

转义字符	简单说明
\a	生成声音提示
\b	退格符
\c	Esc字符
\f	换页符
\n	换行符
\r	回车符
\t	制表符
\v	竖向制表符
\\	反斜线
\'	单引号
\nnn	采用1、2或3位八进制数值表示的等价ASCII字符
\xHH	采用1或2位十六进制数值表示的等价ASCII字符
\cX	Ctrl-X字符

根据上一节的介绍可知，Shell中的元字符都具有特殊的含义，无法直接使用。而为了引用元字符本身，则可以采用三种形式。一种是在元字符前面加转义符号“\”，用以表示字符文字本身，而非具有特殊意义的元字符。

例如，下列两个echo命令的最终结果大不相同：

```
$ echo *
Chkfs chkroot ftpget menu1 menu2 settcp upload
$ echo \*
*
$
```

转义符号“\”是一种引用单个（特殊）字符的最佳方法。一个（特殊）字符前如果增加了转义符号，等于告诉Shell，随后的字符应按普通字符文字解释，例如，“*”、“\\$”、“\\”、“\/”以及“\|”等分别表示“*”、“\$”、“\”、“/”以及“|”等字符本身，参见下面的例子：

```
$ echo "Hello"
Hello
$ echo "\"Hello\", he said."
"Hello", he said.
$ echo "\$var"          # “\$”后面的变量不做解释
$var
$ echo "\\ "
\
$
```

由此可见，为了引用单个字符，使用转义符号“\”是最方便的方法。但当需要引用多个字符时，“\”就显得非常笨拙了。因此，Shell提供第二种引用方式，即采用单引号的方式引用元字符。在此情况下，单引号之间的所有字符（包括元字符）均按普通文字本身解释。例如，为了引用一个字符串“**”，我们可以在“**”前后增加一对单引号：

```
$ echo xx'**'yy
xx**yy
$
```

这意味着，单引号中的所有元字符（反向单引号“`”和上述的转义字符除外）均作为文字常量处理。因此，利用单引号可以同时引用多个元字符或字符串，如下所示：

```
$ echo '*.c $var "testing"'
*.c $var "testing"
$
```

从上述例子中还可以看出，单引号“'”不允许使用美元符号“\$”引用变量的值，其中的\$var仅表示其本身。

在实际应用中，使用单引号引用参数或变量，其作用是防止字符串中的特殊字符被Shell提前解释或扩展。例如，下面的命令表示列出所有以大写字母C或小写字母c为起始字符的文件：

```
$ ls -l [Cc]*
-rwxr-xr-x 1 gqxing gqxing 255 2009-11-18 13:36 Chkfs
-rwxr-xr-x 1 gqxing gqxing 379 2009-11-18 13:36 chkroot
$
```

如果在“[Cc]*”前后增加单引号，则可理解为引用元字符本身，也就是把“[Cc]*”作为一个字符串解释。在下面的例子中，单引号中的“[Cc]*”就是作为一个文件名解释的，因而找不到匹配的文件。


```
$ ls -l '[Cc]*'
ls: cannot access [Cc]*: No such file or directory
$
```

在Linux系统中，某些命令或实用程序会重新解释或扩展使用引号传递的字符串中的特殊字符。使用引号引用参数或变量的一个重要用途就是确保能够把其中含有的特殊字符原封不动地传递给被调用的实用程序，由实用程序进行解释或扩展，例如：

```
$ grep '[Ff]irst' menu*
menu1:# First line should be #!/bin/bash.
menu2:# The first line is just a comment.
$
```

假定存在下列文件，怎样使用元字符匹配并删除“test file”呢？

```
$ ls -l test*
-rw-r--r-- 1 gqxing gqxing 0 Nov  6 15:04 test file
-rw-r--r-- 1 gqxing gqxing 0 Nov  6 15:04 test.file
$
```

如果使用问号“?”或星号“*”，将会误删test.file文件。此时可以使用转义符号加空格的形式匹配“test file”，表示test与file之间的空格并非字段分隔符：

```
$ rm test\ file
$ ls -l test*
-rw-r--r-- 1 gqxing gqxing 0 Nov  6 15:04 test.file
$
```

实际上，也可以使用“rm test"file”或“rm test'/file”（使用双引号或单引号括住空格字符）的命令形式删除名字中间包含空格的文件。

第三种方式是利用双引号引用字符串，防止部分（但并非全部）元字符提前解释。在此情况下，除了“!”、“\$”、“/”、“\”和“{”字符之外，其他元字符均按文字本身处理，如：

```
$ echo "The current working directory of **$LOGNAME** is `pwd`"
The current working directory of **gqxing** is /home/gqxing/script
$
```

表3-5总结了三种元字符引用方式的不同处理效果。

表3-5 三种元字符引用的效果

元 字 符 引用 方式	\	\$	*	?	"	'	`
\	不解释	不解释	不解释	不解释	不解释	不解释	不解释
'	不解释	不解释	不解释	不解释	不解释	不解释	
"	解释	解释	不解释	不解释		不解释	解释

3.8 命令历史

Bash以及其他大多数Shell（如Korn Shell、TCShell、CShell和ZShell等）均支持命令历史机制，以便维护用户输入的命令。Shell的命令历史机制和编辑功能使用户能够重复利用先前输入的命令，提高用户的交互访问能力。利用Shell的命令历史机制，能够不加修改地重复执行先前提交的任何命令，或在先前命令的基础上，经过校正命令中的细小打字错误或稍加编辑后组成一个新的命令，然后再重复执行先前的命令。

命令历史机制主要是由Shell提供的下列内部命令和环境变量实现的：

- `fc`命令用于列出（“-l”选项）、编辑（“-e”选项）或重新执行命令历史文件中记录的命令。
- `history`命令用于列出命令历史缓冲区或文件中记录的命令。
- `HISTFILE`变量用于指定命令历史文件。使Shell能够在停止运行之前把缓冲区中的命令历史记录写入指定的文件，以便在下一次启动时Shell能够读回其中保存的、上一次会话期间执行的命令历史记录。如果`HISTFILE`变量未定义，或定义的文件没有“可写”的访问权限，默认的命令历史文件为`$HOME/.bash_history`。
- `HISTSIZE`变量指定命令历史文件的大小。用于限定当前会话期间需要保存到命令历史文件中的命令数量。如果`HISTSIZE`变量未定义，则文件容量的默认值为500，即保存最近执行的500条命令。
- `HISTFILESIZE`变量指定命令历史文件的大小。用于限定不同会话之间需要保存到命令历史文件中的命令数量。如果`HISTFILESIZE`变量未定义，则文件容量的默认值为500，即保存最近执行的500条命令。

3.8.1 `fc`命令

利用特殊的内置命令`fc`，可以按照命令序号或利用命令的起始字符（或字符串），显示、编辑或运行先前执行的命令。在使用`fc`命令列举命令历史缓冲区或文件中的命令时，可以指定单个命令，也可以指定一个命令范围。

实际上，Shell的命令历史机制主要是由`fc`内置命令实现的。`fc`命令允许用户显示、不加编辑或稍加编辑地重新执行命令历史缓冲区或文件中保存的命令，其语法格式如下：

```
fc [ -e ename ] [ -nlr ] [ first [ last ] ]  
fc -s [ old=new ] [ command ]
```

`fc`命令的第一种语法格式表示从用户输入的命令历史缓冲区或文件中选择指定范围（从`first`到`last`）的命令。范围的上限不能超过`HISTSIZE`变量指定的值。`first`和`last`既可以是一个字符串，用于匹配最近执行的、以给定的值为起始字符串的命令；也可以是命令在命令历史缓冲区或文件中的序号（如果在序号前加一个负号“-”，则表示相对于当前命令的偏移值）。如果未指定`last`，则其默认值为`first`。如果`first`和`last`均未指定，则其默认值分为两种情况：在编辑命令历史缓冲区或文件时仅选择前一个命令；在显示命令历史缓冲区或文件时（使用“-l”选项）选择最近执行的16个命令，即从前一个命令开始回溯16个命令。如果使用“-l”选项，将会在标准输出中列出选择的命令。“-n”选项意味着在列出选定范围的命令时，省略命令在命令历史

文件记录中的序号。“-r”选项意味着由近至远反向输出选定范围的命令。

在第一种命令形式中，如果“-nlr”三个选项均未指定，则调用指定的或默认的编辑器，编辑命令历史缓冲区或文件中的命令。命令的范围由first和last确定。

“-e”选项用于定义在校正或编辑先前的命令时使用的编辑器。如果未指定编辑器的名字，则使用FCEDIT变量的值作为默认的编辑器。如果未设置FCEDIT变量，则使用EDITOR变量的值作为编辑器，否则，最终使用nano作为默认的编辑器。在完成命令的编辑之后，退出编辑器时即可输出并重新执行刚才校正过的命令。注意，如果选择编辑多个命令，在退出编辑器时将会依次执行选择的所有命令。

例如，为了列出命令历史缓冲区或文件中序号为10~20的命令，可以使用下列命令：

```
$ fc -l 10 20
10      . setenv
11      set | grep LANG
12      ls -l
13      cd src
14      vim Makefile
15      vim atmmon.c
16      vim listener.c
17      vim handler.c
18      make
19      atmmon
20      ps -ef
$
```

为了列出最近输入的10条命令，可以使用下列命令：

```
$ fc -l -10
466      ps -ef
467      pstree -p
468      ipcs -m
469      ipcs -q
470      pmap -x 6620
471      ls -l /etc
472      cat /etc/exports
473      vim /etc/exports
474      exportfs -r
475      fc -l 10 20
$
```

为了列出最近输入的以cat为起始字符串的命令，可以使用下列命令：

```
$ fc -l cat
472      cat /etc/exports
473      vim /etc/exports
474      exportfs -r
475      fc -l 10 20
475      fc -l -10
$
```

为了利用vim编辑并执行序号为10~20之间的命令，可以使用下列fc命令：

```
$ fc -e vim 10 20
```

第二种命令形式表示跳过编辑阶段。如果存在“old=new”形式的字符串替换，则由Shell直接执行命令行替换，然后重新执行编辑后的新命令。如果未给出命令，则表示执行之前刚执行的命令。例如，为了重复执行先前的ls命令，可以使用下列fc命令：

```
$ ls -l /etc/profile
-rw-r--r-- 1 root root 497 Oct 29 05:49 /etc/profile
$ fc -s
ls -l /etc/profile
-rw-r--r-- 1 root root 497 Oct 29 05:49 /etc/profile
$
```

为了重复执行先前的第115号命令，可以使用下列fc命令（调用which命令，查询匹配的echo命令）：

```
$ fc -s 115
which echo
/bin/echo
$
```

又如，在列出/home/gqxing/incl目录中的文件之后，如果还想查询与incl位于同一层次的src目录中的文件时，可以使用“incl=src”修正先前的ls命令，然后再执行替换后的ls命令：

```
$ ls -l /home/gqxing/incl
total 8
-rw-r--r-- 1 gqxing gqxing 1682 Nov 18 15:15 atm.h
-rw-r--r-- 1 gqxing gqxing 2490 Nov 18 15:16 event.h
$ fc -s incl=src
ls -l /home/gqxing/src
total 148
-rw-r--r-- 1 gqxing gqxing 16188 Nov 18 14:03 atmcom.c
-rw-r--r-- 1 gqxing gqxing 28648 Nov 18 14:04 atmmon.c
-rw-r--r-- 1 gqxing gqxing 76360 Nov 18 14:05 handler.c
-rw-r--r-- 1 gqxing gqxing 28080 Nov 18 14:06 listener.c
$
```

3.8.2 history命令

内置命令history是fc命令的一个特例（在Korn Shell中，history只是利用fc命令定义的一个命令别名，即“alias history='fc -l'”），用于读取、显示或清除命令历史记录中的命令。例如，为了列出命令历史缓冲区或文件中记录的命令，可以使用下列命令（在Bash中，通常会列出命令历史缓冲区或文件中的所有命令，而在Korn Shell中，仅列出16条命令）：

```
$ history
1      ls -l /etc/profile
2      sudo vi /etc/hosts
.....
65     last
66     history
$
```

为了列出最近执行的10条命令，可以使用下列命令：

```
$ history 10
58      cat /etc/host.conf
59      sudo vi /etc/resolv.conf
60      cd /etc/bind
61      sudo vi named.conf
62      sudo vi named.conf.options
63      sudo vi named.conf.local
64      sudo /etc/init.d/bind9 restart
65      host www.google.cn
66      last
67      history 10
$
```

为了清除命令历史缓冲区中的命令，可以使用下列命令：

```
$ history -c
$ history
1      history
$
```



注意 命令历史机制仅适用于交互式Shell，不能在Shell脚本中使用。

3.8.3 重复执行先前的命令

在Bash中，为了重复执行先前的命令，可以利用感叹号“!”引用机制实现。感叹号“!”表示引用命令历史缓冲区或文件中的命令。为了不加修改地重复执行最近刚执行的命令，可以使用“!!”命令。

例如，为了重复执行刚执行的ls命令，可以使用下列“!!”命令：

```
$ ls -l /etc/profile
-rw-r--r-- 1 root root 497 Oct 29 05:49 /etc/profile
$ !!
ls -l /etc/profile
-rw-r--r-- 1 root root 497 Oct 29 05:49 /etc/profile
$
```

“!string”命令表示重新执行最近执行的，以给定的string为起始字符串的命令。而“!?string[?]”则表示重新执行最近运行的，其中包含给定字符串的命令。因此，如果输入“!gcc”命令，将会再次执行最近输入的以gcc为起始字符串的命令。例如，为了重复执行最近一次执行的uname命令，可以使用下列命令：

```
$ uname -n
iscas
$ !un
uname -n
iscas
$
```

“!n”表示重复执行命令历史缓冲区或文件中的第n号命令。而“!-n”则表示重新执行最近执行的倒数第n号命令：

```
$ !36
date
Wed Nov 18 14:10:30 CST 2009
$
```

另外，利用“!![gs/old/new[/]]”命令，还可以先修正刚执行的命令，然后再执行。即在执行之前，先以给定的字符串new替换先前命令中第一个出现的字符串old，然后再执行校正后的新命令。例如，在执行下述第二个命令之前，Shell首先会以给定的字符串cat替换第一个命令中首次出现的字符串file，然后使用替换后的结果作为新的命令提交Shell执行：

```
$ file /bin/zcat
/bin/zcat: Bourne-Again shell script text executable
$ !!:s/file/cat/
cat /bin/zcat
#!/bin/bash
PATH=${GZIP_BINDIR-'/bin'}:$PATH
exec gzip -cd "$@"
$
```

如果命令行中存在多个匹配的字符串，需要全部替换，可以在“s”之前增加一个字符“g”，表示替换所有匹配的字符串。另外，上述第二个命令行中的最后一个斜线字符可以省略。

在Korn Shell中，可以利用命令别名“r”，重复执行先前输入的最近一个命令。例如，如果仅仅输入一个“r”字符，则意味着重复执行最近刚执行的命令：

```
$ ksh
$ echo Hello again
Hello again
$ r
echo Hello again
Hello again
$
```

为了在Bash中也使用Korn Shell用户已经习惯的“r”命令，可用利用下列fc命令定义一个命令别名，然后即可使用“r”命令：

```
$ alias r='fc -s'

或

$ alias r='fc -e -'
```

表3-6 是对先前介绍的“!”命令的总结。

表3-6 常用的部分“!”命令

命令	简单说明
!	表示引用命令历史缓冲区或文件中的命令（除非后面紧跟着空格、换行符、等号“=”或左圆括号“(”等字符）

(续表)

命令	简单说明
!!	重复执行先前刚执行的命令。相当于输入“! <i>i</i> ”命令
! <i>N</i>	重复执行命令历史缓冲区或文件中序号为 <i>N</i> 的命令
! <i>i</i> - <i>N</i>	重复执行从当前命令位置开始倒计数的第 <i>N</i> 个命令
! <i>string</i>	重复执行最近一次执行的, 以给定的部分字符串 <i>string</i> 为起始字符串的命令
! <i>?string[?]</i>	重复执行最近一次执行的, 包含给定字符串 <i>string</i> 的命令
!! <i>string</i>	引用前一个命令, 附加给定的字符串 <i>string</i> , 然后重复执行组合后的命令
! <i>N string</i>	引用第 <i>N</i> 个命令, 附加给定的字符串 <i>string</i> , 然后重新执行组合后的命令
!#	引用迄今为止已经输入的所有字符
!\$	引用前一个命令的最后一个参数
!![: <i>gs</i>]/ <i>old/new</i> /	重复执行先前的命令。在执行之前, 先以给定的字符串 <i>new</i> 替换命令中出现的第一个或全部(如果“ <i>s</i> ”前面有一个字符“ <i>g</i> ”)字符串 <i>old</i>
^ <i>old</i> ^ <i>new</i> ^	重复执行先前的命令(快速替换形式)。在执行之前, 先以给定的字符串 <i>new</i> 替换命令中第一个出现的字符串 <i>old</i>

Bash等Shell还提供其他一些替换与重复执行命令的形式, 感兴趣的读者可以查阅联机文档有关fc命令的介绍。

3.8.4 命令行的编辑与执行

在Bash中, 用户可以利用Shell提供的命令历史机制和命令行编辑功能, 使用熟悉的编辑器, 如vi (vim) 或emacs等, 对先前输入的命令进行编辑, 从而生成新的命令, 然后提交Shell执行。在调用Bash时, 如果使用了“—noediting”选项, 将会关闭命令行编辑功能。

进入Bash之后, Shell将按照FCEDIT和EDITOR变量, 以及emacs编辑器的顺序确定默认的命令行编辑器。如果未设置FCEDIT变量, Bash将使用EDITOR变量的值作为命令行编辑器, 如果未设置EDITOR变量, emacs将成为默认的命令行编辑器。在emacs编辑模式下, 命令行总是处于输入模式, 可以利用上下箭头键翻阅之前输入的命令(这些命令存储在用户主目录下的bash_history文件或由HISTFILE变量指定的命令历史文件中), 使用左右箭头键左右移动光标, 可插入任何字符, 输入遗漏的数据, 还可以使用退格键删除光标左边的字符, 使用Delete键删除光标所在位置的字符, 编辑命令行。

若想使用vi (vim) 编辑命令行, 可以采用下列任何一种设置方式:

```
$ FCEDIT=vi; export FCEDIT
```

或

```
$ EDITOR=vi; export EDITOR
```

另外, 也可以使用set命令设置默认的命令行编辑器:

```
$ set -o vi
```

一旦采取了上述措施之一, 即可进入vi (vim) 命令行编辑模式。命令提示符所在行(通常是终端窗口底部的最后一行)就像一个只有一行数据的编辑窗口。

无论何时，只要按下Esc键，即可进入vi/vim命令模式。当处于vi/vim命令模式时，可以使用上下箭头键翻阅之前输入的命令，或使用“-”或“+”（“j”或“k”键）上下移动命令历史记录，翻阅期望运行的命令。也可以使用“/”、“?”或C等vi/vim命令前后检索命令，使用左右箭头键左右移动光标；使用编辑命令校正命令行，如使用“x”命令删除单个字符，使用“r”命令替换单个字符，使用“R”命令替换字符串，使用“i（或I）”命令插入字符或字符串，使用“a（或A）”命令附加字符或字符串等。

在完成命令编辑之后，按下Enter键即可执行校正后的命令。

与vi（vim）编辑模式相比，emacs编辑模式使用起来更容易。表3-7给出了两种编辑模式常用的命令行编辑命令。

表3-7 常用的命令行编辑命令

vi（命令模式）	emacs	简单说明
上箭头、减号“-”或k	上箭头或Ctrl-P	获取前一个命令
下箭头、加号“+”或j	下箭头或Ctrl-N	获取下一个命令
左箭头、Ctrl-H或h	左箭头或Ctrl-B	左移一个字符位置
右箭头、空格键或l	右箭头或Ctrl-F	右移一个字符位置
b	Esc B	左移一个字
w	Esc F	右移一个字
退格键（编辑模式）	退格键	删除光标位置左边的一个字符
x	Delete或Ctrl-D	删除光标所在位置的一个字符

3.8.5 命令行补充

利用Linux系统提供的Readline库，Bash还支持命令行补充功能。当输入的命令名、文件名或变量名不完整时，可以使用制表符键“Tab”实现命令的补充，由Bash提供名字的剩余部分，从而节省用户的输入时间，也可以利用这个机制查询和检索记不清的命令。

1. 命令名补充

当输入部分命令名而按下Tab键时，Bash将会按照命令检索路径，搜寻以给定文字为起始字符串的命令。如果找不到匹配的命令，Bash将会发出鸣叫声。如果恰好发现一个匹配的命令，Bash将会自动补充命令名的剩余部分。如果发现多个匹配的命令，在vi（或vim）编辑模式下，Bash不会输出任何信息，如果在emacs编辑模式中，Bash将会发出鸣叫声。再按下Tab键之后，Bash将会显示一系列其前缀与用户输入部分匹配的命令，然后允许用户采用同样的方式继续完成命令的选择。

例如，当输入bz，然后两次按下Tab键时，Bash将会给出11个以bz为前缀的命令：

```
$ bz → (Tab) → (Tab)
bzcat          bzegrep          bzgrep          bzless
bzcmp          bzexe           bzip2           bzmores
bzdiff         bzfgrep         bzip2recover
$ bz█
```


如果继续输入字符c, 接着再按Tab键两次, Bash将会给出两个以bzc为起始字符串的命令。如果接着输入a再按Tab键, Bash将会完成bzcat命令的补充, 因为此时只有一个匹配的命令:

```
$ bzc→(Tab)→(Tab)
bzcat  bzcmp
$ bzca→(Tab)→t ■
$ bzcat ■
```

至此, 命令的补充即告完成, 可以接着输入其他命令选项和参数。

2. 文件名补充

同样, 当用户输入一个文件名的起始部分, 接着按下Tab键之后, Bash将会提供文件名的剩余部分。如果用户提供的文件名前缀足以唯一确定一个文件, Bash将会显示一个完整的文件名。如果存在多个匹配的文件, Bash将会尽可能地补充文件名的后续部分, 直至遇到某个分界点需要由用户做出进一步的选择。

假定src目录中存在两个文件: atm_status.c和atm_statistics.c。当用户输入atm, 接着按下Tab键之后, Bash将会把文件名补充到atm_stat, 然后提示用户做出选择:

```
$ vim src/atm→(Tab) vim src/atm_stat■
```

此时, 用户可以根据文件列表做出适当的选择, 如输入“u”或“i”, 再按Tab键即可完成文件名的补充。但是, 如果接着按Tab键, Bash将会重新刷新屏幕, 同时显示一系列可选的文件供用户做出抉择:

```
$ vim src/atm→(Tab)→(Tab)
atm_statistics.c atm_status.c
$ vim src/atm_stat■
```

当输入足够的信息(如“u”或“i”), 且在按下Tab键之后, Bash将会完成文件名的补充:

```
$ vim src/atm_stati→(Tab)→stics.c
```

至此, 用户可以接着输入其他的命令行参数, 或按下Enter键以执行输入的命令。

3. 变量名补充

如同命令和文件名补充一样, Bash也支持变量名的补充功能。当输入一个变量名的起始部分, 接着按下Tab键之后, Bash将会提供变量名的剩余部分, 例如:

```
$ echo $HO→(Tab)→(Tab)
$HOME      $HOSTNAME  $HOSTTYPE
$ echo $HOM→(Tab)→E
```

4. 配置readline库

使用readline库函数的Bash或其他程序需要读取INPUTRC环境变量指定的文件, 获取初始化信息。如果未设置INPUTRC环境变量, 这些程序将会读取\$HOME/.inputrc文件。使用下列语法格式, 可以设置readline库函数的控制变量, 从而控制readline库函数的处理动作。

```
set variable value
```

表3-8列出了readline库函数的部分控制变量(完整的变量及其说明可以使用“man readline”或“info readline”命令获取), 括号中为控制变量的默认值。

表3-8 readline库函数的部分控制变量

控制变量及其默认值	简单说明
editing-mode(emacs)	用于确定使用哪一个编辑器。如果把变量设置为vi，意味着使用vim模式启动readline库函数；设置为emacs意味着使用emacs模式启动readline库函数。设置这个变量的效果相当于执行"set -o vi"或"set -o emacs"命令
horizontal-scroll-mode(Off)	这个变量如果设置为On，意味着超长的命令行将会延伸到屏幕显示区的右边界之外。当把光标移动到右边界时，将会引起长的命令行依次向左移动，从而能够看到命令行的超长部分。这个变量的默认值为Off，在此情况下，超长的命令行部分将会依次延续到第二行等
completion-ignore-case(Off)	用于确定在匹配和补充命令与文件名时是否区分大小写字母。把这个变量设置为On意味着不区分大小写字母
disable-completion(Off)	用于确定是否允许执行命令或文件名补充。把这个变量设置为On意味着禁止执行命令或文件名补充
expand-tilde(Off)	在执行文件名补充时，如果这个变量设置为On，Shell能够扩展波浪号“~”
match-hidden-files(On)	在执行文件名补充时，如果这个变量设置为On，Shell能够匹配以句点“.”为起始字符的隐藏文件
mark-directories(On)	这个变量设置为Off意味着在执行路径文件名补充时不会在目录名之后附加斜线字符“/”

3.9 命令别名

为了照顾用户的使用习惯和对不同操作系统命令的偏好，简化输入的命令，以及提供默认的命令选项，Bash及其他大多数Shell（如Korn Shell、TCShell、CShell和ZShell等）等均提供一种别名机制，使用户能够定义自己喜欢使用的命令名字，以便在输入别名时，Shell能够替换并执行实际的命令（包括选项）。

除了分号、“&”符号、括号、管道符号、书名符号、换行符、空白字符、元字符、引号、等号，以及变量和命令替换字符之外，命令别名可以由任何任意数量的字符或字符串组成。

对于用户而言，别名机制主要是通过alias和unalias命令实现的。其中，alias命令用于定义和列出用户设置的（包括系统定义的）命令别名。其语法格式简写如下：

```
alias [name[=value]]
```

在上述语法格式中，如果别名定义的value中包含空格等空白字符，value前后应加单引号或双引号。例如，为了使用一个简单的命令别名替代chmod命令，以便增加执行Shell脚本的访问权限，可以使用下列alias命令定义一个命令别名：

```
$ alias cx='chmod 755'
$
```

为了使用一个简单的命令别名替代一个复杂的命令，确保使用的ls等命令总是带有一组基本选项，可以使用下列alias命令定义一个命令别名：

```
$ alias ll='ls -l'
$
```

在Linux系统中，如果没有特意使用set命令设置noclobber特性，当使用cp、rm或mv命令复制、删除或重新命名文件时，如果目标文件与现有的文件同名，有可能会在无意之间覆盖或删除同名的文件。为此，我们可以定义下列命令别名，以便在遇到此类情况时能够提示用户做出选择：

```
$ alias cp='cp -i'
$ alias mv='mv -i'
$ alias rm='rm -i'
$
```

在定义命令别名时，等号右边可以包含多个命令，中间由分号隔开，也可以使用管道符号，把多个命令连接起来，构成一个组合命令。例如，为了计数当前目录中的文件，我们可以定义下列别名（注意，当文件名中包含空格字符时，这种文件计数的结果并不准确）：

```
$ alias cf='ls | wc -w'
$
```

当执行上述别名命令时，即可给出当前目录中的文件计数：

```
$ cf
8
$
```

在输入alias命令时如果未指定参数，alias命令将会列出系统定义以及用户设置的所有命令别名。如果仅仅指定了命令别名，Shell将会列出给定命令别名的定义。例如（其中的egrep、fgrep、grep和ls是系统定义的命令别名）：

```
$ alias
alias cf='ls | wc -w'
alias cp='cp -i'
alias cx='chmod 755'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias ll='ls -l'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
$ alias ls
alias ls='ls --color=auto'
$
```

在定义命令别名时，如果语法格式的value中包含变量，则单双引号的选用是非常重要的。在定义命令别名时如果使用的是双引号，value中的任何变量将会在定义过程中进行变量替换。如果使用单引号，value中的变量在调用命令别名之前不会进行变量替换。下面的例子解释了这一差异。

PWD变量包含当前工作目录的路径名。假定用户gqxing在位于其主目录（/home/gqxing）时利用双引号定义了命令别名dir1：

```
$ echo $PWD
/home/gqxing
$ alias dir1="echo Current working directory is $PWD"
$ alias dir1
alias dir1='echo Current working directory is /home/gqxing'
$
```

在建立上述命令别名时，等号“=”右边的\$PWD变量将会立即执行变量替换。因此，当使用“alias dir1”命令显示dir1命令别名的定义时，其输出结果表示变量替换已经发生。不管在何处执行dir1命令，其显示的当前工作目录都是替换后的/home/gqxing，而非实际的工作目录：

```
$ cd /etc
$ dir1
Current working directory is /home/gqxing
$
```

当使用单引号定义dir2命令别名时，将会防止Shell提前解释\$PWD变量。下列“alias dir2”命令的输出表示dir2命令别名仍然保持未替换的\$PWD变量：

```
$ echo $PWD
/home/gqxing
$ alias dir2='echo Current working directory is $PWD'
$ alias dir2
alias dir2='echo Current working directory is $PWD'
$
```

当使用cd命令改换到其他目录时，dir2命令显示的是正确的工作目录：

```
$ cd /etc
$ dir2
Current working directory is /etc
$
```

unalias命令用于删除已经定义的命令别名，其语法格式如下：

```
unalias [-a] [name ...]
```

其中，“-a”选项用于删除已经定义的所有命令别名，包括系统定义的命令别名。为了删除或撤销某个特定的命令别名，可以在unalias命令中直接指定。例如，为了删除已定义的命令别名ls，可以使用下列命令：

```
$ unalias ls
$ alias ls
ls:
$
```

下列命令将会删除系统定义或用户设置的所有命令别名：

```
$ unalias -a
$ alias
$
```



命令别名不能递归地定义。但是，在别名定义等号右边的别名值（value）中，如果最后一个字符是空格（或制表符），则紧跟命令别名的下一个命令也将做别名检查和替换。此外，使用Shell函数也可以实现别名机制的所有功能。

3.10 作业控制

现在, 几乎所有的Shell均支持作业控制功能。在Bash中, set命令的“-m”或“-o monitor”选项用于启用Shell的作业控制功能。通常, 作业控制功能总是启用的。如果作业控制功能不正常, 可以检查这个设置是否正确。

除了进程ID之外, Shell还会为每个作业分配一个数字较小的作业号。例如, 当利用&符号启动后台作业, 并使之异步运行时, Shell将会输出下列信息, 其中第一个数字表示作业号, 第二个数字是作业的进程ID:

```
$ find / -name "*conf" -print > conf.log 2>&1 &
[2] 2154
$
```

Shell采用作业控制表, 记录和跟踪当前的作业。利用jobs内部命令, 可以显示作业控制表中保存的当前作业。因此, 为了查询系统中的后台作业信息, 可以使用jobs命令列出系统中的所有作业, 其中包括作业号、作业的命令行以及正在运行或暂时停止运行等状态信息。下面的例子说明当前系统中存在两个后台作业, 其中一个作业正在运行, 另外一个作业已处于停止运行状态:

```
$ jobs
[1]+  Stopped      overload.sh
[2]-  Running      find / -name "*conf" -print > conf.log 2>&1 &
$
```

当运行一个较长时间才能完成的程序时, 为了能够在此期间继续执行其他任务, 可以使用Ctrl-Z组合键以及bg命令, 把程序放入后台运行。按下Ctrl-Z组合键时将会向Shell和当前程序发送一个STOP信号。收到此信号之后, Shell将会输出一个表示作业已经停止(“Stopped”)的信息, 接着输出一个命令提示符, 示例如下:

```
$ overload.sh
^Z
[1]+  Stopped      overload.sh
$
```

此时, 用户可以改变作业的运行状态, 或者使用bg命令把作业放到后台运行, 或者在完成其他任务后, 利用fg命令仍让作业回到前台运行。

例如, 为了把停止运行的作业放到后台运行, 可以使用下列命令:

```
$ jobs
[1]+  Stopped      overload.sh
[2]-  Running      find / -name "*conf" -print > conf.log 2>&1 &
$ bg %1
[1]+  overload.sh &
[2]   Exit 1        find / -name "*conf" -print > conf.log 2>&1
$ jobs
[1]+  Running      overload.sh &
$
```

为了让一个后台作业回到前台继续运行，可以使用下列命令：

```
$ jobs
[1]+  Stopped      overload.sh
[2]-  Running      find / -name "*conf" -print > conf.log 2>&1
$ fg %2
find / -name "*conf" -print > conf.log 2>&1
$
```

为了停止一个后台作业，可以使用下列命令：

```
$ jobs
[1]+  Stopped      overload.sh
[2]-  Running      find / -name "*conf" -print > conf.log 2>&1
$ kill %1
[1]-  Terminated  overload.sh
[2]+  Exit 1        find / -name "*conf" -print > conf.log 2>&1
$ jobs
$
```

为了等待一个当前正在运行的作业的完成，可以使用下列命令：

```
$ jobs
[1]+  Stopped      overload.sh
[2]-  Running      find / -name "*conf" -print > conf.log 2>&1
$ wait %2
[2]-  Exit 1        find / -name "*conf" -print > conf.log 2>&1
$
```

通常，后台作业可以输出数据，但不允许从终端读取数据。如果后台作业需要从终端读取数据，作业将会停止运行。另外，如果使用“stty tostop”命令设置终端选项，将会禁止后台作业输出数据。此后，当遇到数据输出或需要从终端读取数据时，后台作业也会停止运行。

除了作业号之外，Shell还提供若干方法，以使用户选择后台作业，详列如下：

- %number：使用作业号引用后台作业；
- %string：使用给定的字符串作为命令行起始字符串引用作业；
- %?string：引用命令行含有给定字符串的作业；
- %%：引用当前作业；
- %+：等价于%%，表示当前作业；
- %-：引用前一个作业。

在Ubuntu Linux系统中，如果用户提交了后台作业，不管作业的当前运行状态如何，一旦使用exit命令、Ctrl-D组合键或关闭终端窗口退出系统，系统都会不加警告地立即终止后台作业的运行。如果想在退出系统后仍然确保后台运行的作业能够继续运行，直至其正常结束，可以利用nohup命令实现。

在此情况下，Shell会随时监控用户提交的后台作业及其运行状态。例如，当提交的后台作业处于停止运行状态而准备退出系统时，用户将会收到一条警告信息：“There are stopped jobs.”。此时，可以利用jobs命令查询后台作业，决定是否需要等待作业的完成。如果经过确认后仍然想要退出系统，而不关心作业是否继续运行，Shell会立即终止已经处于停止运行状态的后台作业，例如：

```
$ nohup overload.sh
nohup: ignoring input and appending output to 'nohup.out'
^Z
[1]+  Stopped                  nohup overload.sh
$ exit
logout
There are stopped jobs.
$ exit
logout
```

但是，如果提交的作业处于后台运行状态，当使用exit命令、Ctrl-D组合键或关闭终端窗口而退出系统时，系统将会不加警告地立即退出系统，但后台作业仍会继续运行，例如：

```
$ nohup overload.sh &
nohup: ignoring input and appending output to 'nohup.out'
[1] 7680
$ exit
logout
```

这可以从另外一个终端窗口中，利用下列命令验证上述事实（overload.sh脚本仍在继续运行）：

```
$ ps -ef | grep overload
gqxing    7680      1  0 14:48 ?        00:00:00 /bin/sh overload.sh
$
```

事实上，当使用nohup命令调用某个命令时，nohup的功能是让调用的进程忽略SIGHUP信号，同时把进程的输出重定向到用户主目录下的nohup.out文件中。当需要提交一个运行时间很长，而用户又不想一直待在终端之前静等命令的最终处理结果时，可以借助于nohup命令。

在退出Shell时，系统将会向用户注册Shell的所有子进程发送一个SIGHUP信号。通常，这个信号将会引起用户的所有进程终止运行。如果使用nohup命令，所有正在运行的进程或后台作业，包括本身已经屏蔽了SIGHUP信号的程序，将会忽略SIGHUP信号而继续运行。

3.11 会话记录与命令确认

3.11.1 保存会话记录

Linux系统提供一个很有用的工具——script，通过scrip命令，可以记录一个用户从注册到退出系统的整个或部分会话过程，其中包括用户的输入和系统的响应信息。实际上，这个工具仅适用于字符终端设备，尽管也能够捕捉vim编辑器等会话过程，但由于在vim的编辑过程中很可能会使用控制字符，执行诸如光标移动等操作，使得捕捉的内容难于阅读，如果利用cat命令显示捕捉的vim会话过程有时很可能会一闪而过，因而来不及阅读，致使记录的内容意义不大。因此，通常最好使用编辑器（如vim等）等工具查阅会话过程的记录文件。

通常，script命令捕捉的会话过程将会记录在当前目录下的typescript文件中。为了使用不同的文件存储会话过程，可以在script命令之后指定一个不同的文件名。为了依次记录用户的每一个注册会话过程，可以使用“-a”选项。否则，script将会覆盖原有的文件内容，删除先前的会

话记录。下面的例子说明了怎样使用script命令记录用户的会话过程:

```
$ script
Script started, file is typescript
$ date
Thu Nov 19 01:03:14 CST 2009
$ uname -r
2.6.31-14-generic
$ uptime
01:03:58 up 13 min,  2 users,  load average: 0.00, 0.27, 0.36
$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 169.254.78.100
    netmask 255.255.255.0
    gateway 169.254.78.1
$ exit
exit
Script done, file is typescript
$
```

为了终止会话记录功能, 可以使用exit命令退出script (这里的exit命令并不是退出Linux系统)。之后, 可以使用cat、less、more或编辑器查阅typescript文件。下面的例子就是利用cat命令查阅刚才创建的typescript文件时的结果:

```
$ cat typescript
Script started on Thu Nov 19 01:03:08 2009
$ date
Thu Nov 19 01:03:14 CST 2009
$ uname -r
2.6.31-14-generic
$ uptime
 01:03:58 up 13 min,  2 users,  load average: 0.00, 0.27, 0.36
$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 169.254.78.100
    netmask 255.255.255.0
    gateway 169.254.78.1
$ exit
exit

Script done on Thu Nov 19 01:08:35 2009
$
```



当使用vim、emacs或其他编辑器编辑typescript文件时, 可以使用Ctrl-V键、Ctrl-M键和“s”等编辑命令删除多余的回车字符, 参见第6章中的介绍, 也可以利用dos2unix或tr命令消除文件中每一行后面附加的回车字符“^M”。


```
$ dos2unix -n typescript newfile
$
```

或

```
$ cat typescript | tr -d '\r' > newfile
$
```

在上述tr命令中，“-d”选项意味着删除随后指定的转义字符“\r”，也即可见的回车字符“M”。此外，如果想用dos2unix命令，需要事先安装tofrodos软件包。

3.11.2 命令的查询与验证

1. which命令

当输入一个Linux命令时，Shell将会按照PATH变量列出的目录顺序，也即所谓的检索路径，依次检索与之匹配的命令。如果用户改变了PATH变量定义的检索路径的顺序，Shell究竟执行的是哪一个目录下的同名命令，则不得而知。

为此，可以使用Linux系统中的which命令予以验证。例如，当用户对命令的运行结果超出预期的结果而感到怀疑时，为了弄清当前究竟执行的是哪一个tar命令，可以使用下列命令：

```
$ which tar
/bin/tar
$
```

通过运行which命令，可以找出当前究竟执行的是哪一个目录下的同名命令。有关系统目录的说明，参见第4章“文件系统基础知识”。注意，which命令并不考虑内置命令，它仅检索并给出发现的第一个外部命令。

2. whereis命令

whereis是另外一个具有类似功能的命令，用于检索与给定命令相关的文件（根据标准的目录位置，而非检索路径）。例如，下列命令可用于寻找与tar命令有关的文件及其位置：

```
$ whereis tar
tar: /bin/tar /usr/include/tar.h /usr/share/man/man5/tar.5.gz /usr/share/man/man1/tar.1.gz
$
```

在上述例子中，whereis发现了三个与tar有关的文件：tar命令本身、tar头文件和tar的联机文档。

3. which与whereis命令的比较

当给定一个命令或程序的名字时，可以使用which命令，按照命令检索路径变量PATH设定的目录及其顺序，检索究竟执行的是哪一个命令或程序。在设定的命令检索路径中，如果给定名字的命令或程序不止一个，which命令将会显示第一个发现的命令或程序，也就是最有可能调用的命令或程序。

对于给定的命令或程序，whereis将会采取一种独立于检索路径的方式，按照Linux系统中的一系列标准目录进行检索，找出相应的二进制程序文件、联机帮助手册和程序的源代码，以及所有与之相关的文件。

4. apropos命令

当需要执行某个特定的处理任务，但又不知道确切的命令名字，甚至根本就不知道使用什么命令时，可以借助于apropos命令，利用关键字检索可用的命令。apropos将会利用提供的关键字，检索所有手册页中的命令简述部分，找出匹配的命令。实际上，apropos与“man -k”命令的功能完全一样。

下列例子表示，当利用who作为关键字运行apropos命令时，系统将会给出一系列与之相关的命令，说明每个命令所属的手册类型，以及命令的简单描述，而这个描述即取自命令手册页前面的简要说明部分。

```
$ apropos who
at.allow (5)      - determine who can submit jobs via at or batch
at.deny (5)       - determine who can submit jobs via at or batch
from (1)          - print names of those who have send mail
w (1)             - Show who is logged on and what they are doing.
w.procps (1)      - Show who is logged on and what they are doing.
who (1)           - show who is logged on
whoami (1)        - print effective userid
whois (1)         - client for the whois directory service
$
```

5. whatis命令

与apropos命令可以执行模糊检索相比，whatis命令只能检索与给定关键字完全匹配的命令。例如：

```
$ whatis who
who (1)          - show who is logged on
$
```

第4章 文件系统基础知识

文件系统是Linux操作系统的重要组成部分之一，承担信息处理的组织、管理和维护等任务。而文件则是Linux系统中信息存储、读写和执行的基本单位。操作系统正是通过文件系统管理信息的存储、传输和加工等多种处理功能的。本章主要介绍文件系统的基础知识，讨论文件系统的层次结构和组织结构，介绍Linux系统支持的各种文件类型，以及文件的安全保护机制。

4.1 文件系统的层次结构

在Linux系统中，信息的基本组织单位称做文件。Linux文件系统采用一种逻辑的方法组织、存储、访问、操作和管理信息。把文件组织在一个层次目录结构的文件系统中，每个目录包含一组相关文件的组合。Linux系统的一个重要特性是提供一种通用的文件处理方式，简化物理设备的访问；按文件方式处理物理设备，允许用户以同样的命令处理普通文件和物理设备。例如，在打印机上打印文件与在终端屏幕上显示文件的处理方式是类似的。

4.1.1 树形结构

从理论方面讲，文件系统是文件的一种逻辑组织结构。从用户的角度看，Linux的文件系统只是一个按层次结构组织的目录文件树，文件系统的起点是根目录root。根目录相当于整个目录文件树的根，如图4-1所示。

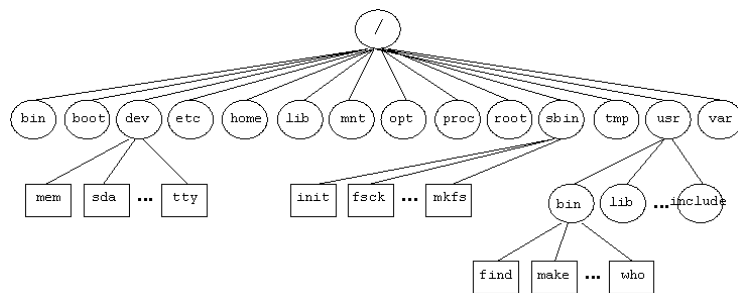


图4-1 目录文件的层次组织结构

子目录是整个目录文件树形层次组织结构中的一个中间节点，是比当前目录层次低一级的目录。文件是整个目录树形层次组织结构中的一个叶子节点。例如，如果/usr目录是当前目录，那么所有位于/usr下面的目录及其子目录都是当前目录的子树。如bin和lib就是/usr下边的子树。除非明确指定了目录路径，大多数Linux系统命令均把文件参数看做当前目录中的文件。

在文件系统中，若干文件可以组成一个目录，而若干目录则可以构成一个目录的层次组织结构，而位于目录层次结构顶端的就是一个称之为根目录的特殊目录，根目录包含了各种系统目录和文件，例如 /bin、/boot、/dev、/etc、/home、/lib、/proc、/sbin、/tmp、/usr以及/var等标准目录。图4-1给出的是一个简化的文件系统层次组织结构。

在操作系统中，文件系统的设计目的就是要把文件有序地组织在一起。Linux系统提供一种便于用户从逻辑上组织文件的文件系统。

Linux系统鼓励用户按照一定的原则建立目录，例如，存储源程序的目录、存储目标程序的目录，以及存储文档的目录等。Linux文件系统的关键思想是其层次组织结构，不管系统中拥有多少用户，每个用户都可以创建若干目录及其子目录，分类存储自己的文件。

另外，文件的访问权限是多用户计算机文件系统的一个重要组成部分，用于保护用户的数据安全。

Linux系统的一个重要特性是，所有的I/O设备都与特殊文件联系在一起，用户无需了解硬件设备的读写方式，只需像操作普通文件一样操作特殊文件，即可达到访问I/O设备的目的。例如，读取特殊文件相当于从硬件设备中直接读出数据，写特殊文件则相当于直接向硬件设备发送数据。利用特殊文件，实现了用户程序与硬件设备之间的通信，由文件系统管理硬件设备的I/O处理，使硬件设备对用户是透明的。

4.1.2 路径名

无论何时，当前工作目录中的所有文件都是可以直接存储的。通过名字，可以直接引用文件。而对于非当前目录中的文件，必须在文件名之前加上各级目录路径才能访问。文件的路径名指的就是从某个目录（如根目录或当前目录）开始，穿过整个文件系统，直至到达目标文件而经过的一目录层次路径。例如，从根目录（/）开始，中间经过usr和bin两级子目录的一条路径就是find文件的路径名：

```
/→usr→bin→find
```

把上述访问路径写成Linux文件系统中的标准路径名就是/usr/bin/find。

文件的访问路径可以有两个起点：一是从当前工作目录开始；二是从根目录开始。凡是以根目录为起始位置，即以斜线字符“/”为起始字符的路径名称为绝对路径名。绝对路径名指定了文件在文件系统的层次组织结构中从根目录开始的存储位置。

相对路径是指相对于当前工作目录而言的目录。凡以当前工作目录或其他以非斜线字符为起始字符的所有路径名都是相对路径名。相对路径名指定了文件在文件系统中相对于当前工作目录的存储位置。

例如，路径名/usr/include/stdio.h就是一个从根目录开始的绝对路径名。其中，usr是根目录的子目录，include是usr目录的子目录，stdio.h是这个目录层次末端的一个文件。

include/stdio.h是相对于usr目录的一个相对路径名，而stdio.h则是相对于usr/include目录的一个相对路径名。

此外，每个目录中均包含以句点“.”和双句点“..”命名的两个特殊目录文件，分别表示当前目录及其父目录。这两个特殊目录把文件系统各级目录有机地联系在一起。其中句点“.”是当前目录的别名，凡是期望访问当前目录中的文件时，都可以直接使用句点“.”而不必明确给出当前目录名。双句点“..”是当前目录父目录的别名。从任何目录位置开始，使用双句点“..”形式的父目录，可以逐层攀升到文件系统层次组织结构的最上层。

下面几个简单的规则适用于所有的路径名：

- 如果路径名以斜线字符开始，则说明路径名是从根目录开始的绝对路径名，除此之外，其他所有的路径名都是相对于当前目录的相对路径名。

- 路径名或者是由斜线字符分隔的一系列名字，或者是单个名字。在一串名字中，最后一个名字就是实际的文件，其他名字均为目录。文件可以是任何类型的文件。
- 在任何目录位置，在路径名中使用双句点“..”符号可以往上攀升文件系统的目录层次。在路径名中，除了双句点“..”之外的其他所有名字均为降低目录层次。

4.2 文件系统的组织结构

在Linux文件系统中，目录和文件的组织结构是有一定规律可循的。这种有序的组织结构有助于用户访问、管理和维护Linux系统，图4-2给出的是一个典型的文件系统目录组织结构。

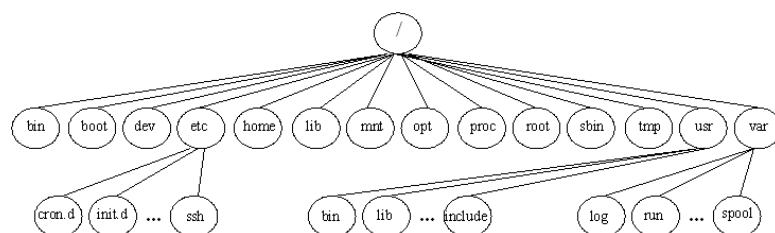


图4-2 Linux文件系统的典型目录结构

表4-1是对Linux系统部分主要目录的简要说明。

表4-1 Linux系统的部分主要目录

常见目录	典型子目录	说明
/bin		其中包含系统、系统管理员和普通用户可以共享的各种通用程序，如cat、cp、mv、rm、ls、ps等常用的基本命令，以及各种Shell，如bash等
/boot		其中包含系统引导程序、Linux内核程序文件vmlinuz、磁盘内存映像文件initrd及GRUB初始引导程序和配置文件等
	grub	其中存有GRUB配置文件，及三种不同类型的初始引导程序等
/dev		在Linux系统中，任何设备均对应一个或多个特殊文件（也称设备文件），这个目录包含系统支持的所有设备文件。例如，console表示系统控制台，lp0表示打印机，ttyXX表示串口设备，cdrom表示CD/DVD驱动器，dsp文件表示音响设备，写到这个文件的任何数据，将会重定向到音响设备。mem表示物理内存，kmem表示系统内核占用的虚拟内存，sda表示连接到主控制器的第一个磁盘，sda1和sda2等则分别表示其中的第一和第二个磁盘分区等
/etc		这个目录是整个Linux系统的中心，其中包含所有系统管理和维护方面的配置文件，如host.conf、inetd.conf、logrotate.conf、mke2fs.conf、nsswitch.conf、pam.conf、resolver.conf、sysctl.conf、syslog.conf和vsftpd.conf等，此外，还有其他大量的配置文件分别位于单独的子目录中。通常应注意备份这个目录中的重要配置文件，以便需要时能够快速恢复系统
	apache2	Apache配置文件的根目录，其中包含Apache服务器的各种配置文件，如apache2.conf等。Apache是一个通用的、高性能的HTTP服务器，也是世界上最流行的Web服务器。Apache采用模块化的设计方式，支持运行时的动态模块选择、虚拟主机，以及服务进程数量的动态调整等
	apt	其中包含软件管理工具使用的配置文件，如sources.list等

(续表)

常见目录	典型子目录	说明
	bind	其中包含named域名服务守护进程使用的各种配置文件，如named.conf等
	cron.d	用于存储cron进程调度运行后台进程所用的配置和控制文件。其他有关的目录包括cron.hourly、cron.daily、cron.weekly和cron.monthly等4个目录
	cups	用于存储通用UNIX打印系统（Common UNIX Printing System, CUPS）使用的各种配置文件
	default	其中的文件用于提供部分工具软件使用的变量及其默认值
	gdm	用于存储GNOME显示管理器的配置文件，如gdm.conf，其中定义了是否允许使用root注册
	init.d	用于存储系统启动过程中需要由init调度执行的脚本文件
	modprobe.d	其中包含系统加载的软件模块的配置文件，如aliases等
	mysql	其中包含MySQL数据库的配置文件，如my.cnf等
	network	其中包含网络接口的配置文件interfaces，及相关的配置工具
	pam.d	其中包含各种用户认证方法的配置文件
	ppp	用于存储PPP的脚本文件和配置文件
	profile.d	用于存储/etc/profile使用的辅助初始化脚本文件
	rcN.d	用于存储进入相应运行级时需由init进程调度执行的脚本文件（其中的N为0、1、2、3、4、5、6或S，表示系统的运行级）
	samba	Samba配置文件的根目录。Samba是一个网络共享软件的总称，Linux系统中实现的SMB协议，允许Linux系统为Windows系统提供文件和打印共享服务
	security	用户存储的基本安全控制文件，包括注册控制文件、控制台访问控制文件，以及资源限制控制文件等
	skel	用于存储最基本的用户初始化文件，如bash_logout、bashrc和profile等。每当增加一个新用户时，都会把其中的部分基本初始化文件复制到用户的主目录中。注意，上述文件均为隐藏文件
	ssh	OpenSSH网络服务所用配置和控制文件的根目录，其中含有sshd_config等重要配置文件
	X11	其中包含X服务器使用的各种配置文件，如xorg.conf等
/home		用户主目录的根目录。每当增加一个新用户，系统将会在/home目录中创建一个形如/home/\$USER的子目录，作为用户的主目录，其中的\$USER为用户名
/lib		这个目录含有系统引导过程，以及运行系统命令所需要的内核模块和各种动态链接共享库文件（扩展名为.so，相当于Windows系统中的.dll文件）。其中，内核模块（驱动程序）位于/lib/modules/kernel-version子目录中
/lost+found		每个文件系统分区都存在一个lost+found目录，用于存储fsck命令在检测与修复文件系统时删除的文件或目录。Linux系统要求正常地关机。一旦由于电源等硬件故障或系统软件故障致使系统异常停机，将会导致文件系统损坏，在重新启动系统时，系统需要使用fsck命令对文件系统进行检测与修复。在检测文件系统期间，fsck将会尝试修复任何受损的文件。部分恢复的文件或无法恢复的数据块将会放置在每个文件系统的lost+found目录中。如果这个目录中存有受损的数据文件，可以使用debugfs等工具，尝试自行恢复丢失的数据。如果是系统文件，也许需要重新安装相应的软件包

(续表)

常见目录	典型子目录	说明
/media		移动存储介质的安装点。当利用GNOME界面安装移动存储介质时，系统将会自动地把移动介质安装到此目录下的某个子目录中
/mnt		文件系统的临时安装点。这是一个通用的安装点，可以临时安装任何文件系统或远程资源。如果愿意，也可以在此目录中创建若干子目录，以便安装不同的设备，如使用/mnt/cdrom和/mnt/usb分别安装CD/DVD和U盘等
/opt		应用程序等附加软件的安装目录
/proc		进程文件系统proc的根目录，其中的部分文件分别对应当前正在运行的进程，可用于访问当前进程的地址空间。/proc是一个非常特殊的虚拟文件系统，其中并不包含“实际的”文件，而是可用以引用当前运行系统的系统信息，如CPU、内存、运行时间、软件配置以及硬件配置等信息。通过读取或修改/proc/sys目录中的适当文件，可以显示或改变运行系统的可调内核参数，相当于执行sysctl命令。最具特色的是，除了kcore等少数几个文件之外，/proc目录中绝大部分文件的大小均为0。以数字命名的目录对应于一个实际的进程，而这个数字则表示进程的ID。这些大小为0的文件相当于一个指向内核数据空间的指针，可据以查询相应的进程信息。由于这个原因，可以把/proc文件系统看做系统内核的一个控制和信息中心。事实上，许多系统命令只是直接显示这个文件系统的一部分文件内容，例如，运行lsmod命令与使用cat命令显示/proc/modules文件的内容，最终的结果是一样的。再如，执行lspci命令与显示/proc/pci文件内容的结果也是一样的
	net	其中的文件分别表示各种网络协议（如TCP、UDP以及ARP等）的状态与统计信息
	sys	这个目录不仅存有各种系统信息，而且也包含系统内核与TCP/IP网络等的可调参数。其中的kernel子目录含有共享内存和消息队列的可调参数，net子目录中含有TCP/IP的各种可调参数。例如，shmmax文件中含有系统的最大共享内存定义，如果使用“echo somevalue >/proc/sys/kernel/shmmax”命令，可以直接修改运行系统的内核参数，而无需重新引导系统。但这一做法需要慎重，有的文件可能包含多个数值，或不同类型的数值，因此，在修改以前一定要弄清参数的意义和实际的数据。为了在每次启动系统时都能使用定制的系统可调参数，可以设置sysctl.conf配置文件，或编写自己的Shell启动脚本，参见第11章的介绍
/root		超级用户root的主目录（在Linux系统中，“/”是整个系统的根目录，而非超级用户的主目录）
/sbin		其中包含与系统引导、管理和维护，以及与硬件配置等方面的有关命令或脚本文件，如fdisk、init和lilo等。在安装了/usr等文件系统之后才需要执行的系统命令通常放置在/usr/sbin目录中。注意，/sbin目录中的命令主要供超级用户使用，普通用户通常无法使用
/srv		用于存储本地系统提供的服务进程所用的数据文件（现为空目录）
/sys		系统各种设备配置信息的根目录。例如，block子目录中含有磁盘及磁盘分区的配置信息，bus子目录中含有pci和usb等的配置信息和驱动程序
/tmp		临时文件目录。用于存储系统运行过程中生成的临时文件，也可以供用户存储自己的临时文件。在系统的运行过程中，许多程序均使用这个目录存储临时数据文件，其中的许多文件对于当前正在运行的进程而言是非常重要的，删除这样的文件可能会导致系统瘫痪。因此，一般不要自己删除这个目录中的文件。

（续表）

常见目录	典型子目录	说明
/usr		除非确实有把握。在大多数系统，尤其是在UNIX系统中，伴随着系统的启动过程，将会清除这个目录中的所有文件
		/usr既可以作为一个单独的文件系统，也可以作为根目录下的一个子目录，其中存有系统提供的各种共享数据（如用户命令、库函数、头文件和文档等）
	bin	其中包含用户经常使用的各种命令
	games	其中包含游戏和教育等方面的程序
	include	用于存储各种C语言头文件。这个目录及其子目录中的头文件是C语言程序开发人员需要经常引用的文件。其中，sys、linux和bits等子目录中定义的数据结构，对于深入学习、理解和掌握Linux系统具有极大的参考价值
	lib	其中包含各种共享的库函数，可供程序员以静态或动态方式链接自己开发的应用程序
	sbin	其中包含系统引导完成之后系统管理员经常使用的各种系统管理和维护命令
	share	共享目录，其中含有man（联机文档的根目录）、info（GNU info文档的根目录）、doc（各种软件包特定的文档）、locale（语言环境）、vim（用户指南）以及zoneinfo（时区定义）等子目录
	src	用于存放Linux系统内核的源代码、头文件、Makefile和文档等
/var		/var既可以作为一个单独的文件系统，也可作为根目录下的一个子目录，用于存储各种可变长的数据文件（如日志文件）、暂存文件或待处理的临时文件等
	cache	APT和samba等程序使用的工作目录。用于缓存程序使用的各种数据文件，尤其是缓存APT软件工具下载的软件包和信息文件
	lib	用于存储软件包特定的动态链接共享库、配置文件、数据文件和状态信息等
	lock	用于存储各种服务进程或应用程序访问特定的设备或文件时设置的封锁文件
	log	系统守护进程日志文件的储存目录，其中包括lastlog（每个用户最后一次注册的时间记录）、messages（由syslogd记录的所有内核和系统程序的日志消息）以及wtmp（所有用户的系统注册/注销记录）等重要文件。位于/var/log目录中的文件会不断地增长，因而要求定期地备份或清除。通常，Linux系统均采用以日、周或月为时间周期，定时执行例行检查，以循环截取（如使用/usr/sbin/logrotate一类的程序）的方式，删除过时的数据，保留一定时间范围的最新数据，使文件的大小保持一个适中的规模。在Ubuntu Linux系统中，每日将会定时执行一次logrotate程序，检查并处理系统日志文件。参见/etc/cron.daily/logrotate文件
	mail	这个目录存有每个用户电子邮件的邮箱文件，其中的每个文件均以用户的用户名命名
	run	系统运行信息文件的根目录，其中的各种pid文件存有相应守护进程的PID，另外一个典型的文件是/var/run/utmp，其中含有当前系统中的用户注册信息
	spool	用于缓存各种等待处理的文件，如打印任务等。通常，每一类待处理的缓存文件均位于各自的子目录中，如/var/spool/cups等
	tmp	用于存储各种临时文件
	www	Apache服务器的用户文档根目录，用于存储和发布各种HTML文档

4.3 文件的类型

从理论上讲,文件是由一系列连续的字节流组成的,最后以一个EOF字符结束。但从物理实现来讲,文件实际上是由磁盘(或其他存储介质)上的一系列数据块组成的,而组成文件的数据块,并不一定是连续的。

Linux系统可以同时支持许多不同类型的文件系统,以及不同类型的文件。本章所述的文件仅指Linux文件系统支持的基本文件。也就是说,所谓文件乃指驻留在本地磁盘上的,用户经常与之打交道的各种基本文件。

在Linux操作系统中,文件是Linux操作系统中的基本数据组织单位,所有的输入输出操作都是通过文件实现的,系统处理的任何设备和数据均可以归结为对文件的操作。从理论上讲,能够读写普通文件的任何程序都可以读写任何I/O设备。

从用途方面划分,一个文件可以是:

- 文档——也即普通的文本文件,包括脚本、程序源代码、配置数据和日志等。
- 命令——大多数命令都是可执行的二进制数据文件。也就是说,命令是可以执行的程序文件。例如, date命令就是一个可以执行的程序文件,其功能是输出和设置当前的系统日期和时间。
- 目录——简言之,目录是一个包含文件名列表的数据文件。
- 设备——包括终端、磁盘、CD/DVD、磁带机和打印机等。

在Linux系统支持的各种文件系统中,如最流行的Ext2/Ext3文件系统,通常均支持普通文件、目录文件、特殊文件、链接文件、符号链接文件,以及管道文件等6种类型的常规文件(套接字文件不在本章讨论之列),现分述如下。

4.3.1 普通文件

在Linux系统中,普通文件简称文件。所谓文件,实际上是一个命名的数据集合,是一组信息的基本存储单位。通常,每个文件都拥有一个名字,通过名字,可以对文件的数据内容进行处理。

在早期的Minix文件系统中,文件名最多不能超过14个字符。而在Ext2/Ext3等文件系统中,文件名可以长达255个字符。在不同的目录中,文件可以同名,但在同一个目录中,则不允许同名的文件存在。原则上,文件名可由任何字符组成,但若包含不可打印的字符、空白字符及Shell元字符,在实际应用过程中将是非常麻烦的。

普通文件可以存储任何数据,存储的内容可以是ASCII文本、源代码、Shell脚本、配置数据及各种文档等,也可以是二进制程序代码等。由此可见,普通文件是Linux文件系统中应用最多的一种文件类型。

此外, Linux系统中的文件并没有所谓的记录、结构以及内容之分,所有文件都是由一维的字符流组成的,存储在磁盘等存储设备中,在文件系统中占有零个或多个数据块。而且,数据也并非一定驻留在连续的磁盘块中。但操作系统隐藏了这一内部存储形式,用户看到的是一系列连续的字节流。对于用户而言,无需考虑文件的底层存储结构。

当然,也可以根据数据内容把文件分类为C源代码、Shell脚本、文本数据以及二进制可执

行程序等。但Linux系统却不这样认为，对于存储不同数据的普通文件，Linux操作系统不加任何区别，也不要求一个普通文件必须采用什么记录结构。

但是，为了便于处理与维护，Linux系统也定义了若干标准数据文件格式，例如，二进制可执行文件必须采用a.out文件格式，一些档案文件前部通常都会包含一个文件格式代码（magic code），用于标识文件的内容。例如，cpio命令生成的档案文件，其标识代码为070707等（参见/etc/magic文件）。此外，应用程序也可以按一定的记录格式组织和存储自己的文件。实际上，数据库等应用程序也是这样组织和使用文件的。

由于Linux系统并不刻意区分文件的类型，不像Windows系统那样，以扩展名区分文件的类型。因此，单从文件名本身来看，Linux系统中的大部分文件都无从知道其类型。Linux文件系统也没有对文件的命名强加任何约定，但由于应用程序的需要及用户的使用习惯，通常以“.c”作为C源程序文件名的后缀，以“.sh”作为Shell脚本文件名的后缀。按照惯例，可执行程序的文件名通常不加任何后缀。

当需要确定文件的内容，判断文件的类型时，可以选用一定的工具，其中最典型的就是ls和file命令。例如，下列file命令用于确定指定文件的内容与类型（实际上，标识代码也是file命令用于确定文件内容与类型的方法之一）：

```
$ file /bin/gunzip
/bin/gunzip: Bourne-Again shell script text executable
$ file /etc/mo*
/etc/modprobe.d:      directory
/etc/modules:         ASCII English text
/etc/mono:            directory
/etc/motd:            symbolic link to `/var/run/motd'
/etc/motd.tail:       ASCII text
$
```

从用户的角度来看，普通文件可以分为两种类型：即文本文件和二进制数据文件。文本文件只包含可打印字符，如ASCII字符、中文字符等，而二进制数据文件中的每个字节允许有256种数值。

在Linux文件系统中，通常提供下述文件操作：打开文件（open）、创建文件（create）、读文件（read）和写文件（write）等。Linux系统提供大量的文件操作命令，用于创建、编辑和处理文本文件。例如，为了显示一个文本文件，可以使用下列cat命令（有关文件的各种操作与处理方法，详见第5章）：

```
$ cat /etc/issue
Ubuntu 9.10 \n \l
$
```

对于二进制数据文件来说，其中的内容无法直接显示，因为每个字节的256种取值大部分都是不可打印字符。若想显示（而不是执行）二进制数据文件的内容，可以使用专门的命令，如od命令，以ASCII字符、八进制数据或十六进制数据等形式显示二进制数据文件。例如，若想检查一个二进制数据文件的内容，尤其是头部是否包含特定的标识代码，可以使用下列命令：

```
$ od -c cpiofile
0000000  0  7  0  7  0  7  0  0  4  0  0  7  1  5  5  0
```

```

0000020 2 1 0 4 0 7 5 5 0 0 1 7 5 0 0 0
0000040 1 7 5 0 0 0 0 0 0 2 0 0 0 0 0 0
0000060 1 1 1 1 6 4 2 7 6 5 5 0 0 0 0 0
0000100 7 0 0 0 0 0 0 0 0 0 0 0 s c r i
0000120 p t \0 0 7 0 7 0 0 0 4 0 0 7 1
.....
$

```

4.3.2 目录文件

目录文件简称目录。目录也是一种文件，是一种特殊类型的文件，其中存储的内容不是普通意义上的数据，而是一系列文件名及其信息节点号。除了存储的内容不同之外，目录文件与普通文件在文件系统中的存储方式是一样的。目录用于提供文件名、信息节点与文件数据之间的关联关系。因而可以说，目录的结构也决定了文件系统的组织结构。

严格地讲，目录文件是由一系列目录项组成的，而每个目录项又主要是由两个不同的字段组成的：一个字段为信息节点号，用于引用信息节点，另一个字段为文件的名称。“ls -ai”命令可用于列出目录中的文件名及其信息节点号，下面是这个命令的输出结果（其中第一列为信息节点号，第二列为文件名）：

```

$ ls -ai | sort -n
2 .
2 ..
11 lost+found
12 etc
16 cdrom
17 bin
18 dev
.....
$

```

在目录文件中，每个目录项通过信息节点号实现了文件名与文件数据的映射。通过文件名可以找到其信息节点，通过信息节点最终又可以找到文件中的实际数据内容。在上述的例子中，文件名/lost+found的相应信息节点号是11，是继根目录（/）和保留的信息节点位置（用于日志文件）之后创建的第一个目录。

如前所述，Linux文件系统中的每个目录均包含两个特殊的目录，即以句点“.”和双句点“..”表示的当前目录及其父目录。因此，为了进入当前目录或父目录，用户可以相应地引用“.”或“..”。例如，如果当前的工作目录为/usr/include/sys，输入“cd ..”命令后即可进入/usr/include目录，如下所示：

```

$ pwd
/usr/include/sys
$ cd ..
$ pwd
/usr/include
$

```

“usr”是一个目录文件，虽然其中包含的数据与普通文件并无实质的差别，但与普通文件不同的是，目录文件是由Linux文件系统直接管理的，用户只能查询其中的文件列表，不能

使用编辑器等工具直接修改目录文件，也就是说，用户只能读取目录文件，只有操作系统才能写目录文件。

尽管超级用户可以利用Linux系统提供的若干实用程序维护目录文件，但普通用户绝对不能直接写目录文件。例如，超级用户可以利用文件系统调试器debugfs，直接操作文件系统的任何组成部分。但应当了解的是，用于维护目录（或文件系统）的大多数实用程序都是具体文件系统特定的。例如，debugfs主要适用于Ext2/Ext3类型的文件系统。

为了保证目录的完整性，操作系统不允许用户直接修改目录文件。用户可以在目录中创建文件，由操作系统把文件加到目录中。当用户删除文件时，由操作系统从目录文件中删除相应的文件名，目录文件始终是由操作系统负责维护的。

这是因为目录文件毕竟不是普通文件，如果不了解目录的组织结构，不了解文件系统的工作原理，并且处理不当，将会造成无法预料的后果。即使熟悉文件系统，也不能直接操作目录文件。因为目录操作涉及一系列内部的处理步骤和过程，不是一般用户所能胜任的。

在任何目录中，可以存储普通文件、管道文件、特殊文件、符号链接文件，也可以创建目录文件，称做子目录。为了创建一个目录，可以使用下列命令：

```
$ mkdir dirname
```

为了进入不同的目录，可以使用下列命令：

```
$ cd dirname
```

在系统访问期间，用户所在的目录称为当前目录、工作目录或当前工作目录。若想了解自己当前所处的目录位置，可以输入pwd命令。例如，下列命令的输出结果表示用户的当前工作目录为/usr/include：

```
$ pwd
/usr/include
$
```

在Linux系统中，每个用户都拥有一个属于自己的目录，称做用户主目录（或简称主目录）。一旦注册到Linux系统，系统将会自动地把用户引入其主目录。在Linux系统中，用户主目录具有下列形式：

```
/home/username
```

无论何时，通过输入不加任何选项的cd命令，任何用户均可返回自己的主目录。例如，无论在何时何地，一旦输入下列cd命令即可返回作者自己的主目录：

```
$ cd
$ pwd
/home/gqxing
$
```

用户可以在主目录中创建自己的文件，创建自己的子目录。在与Linux系统交互会话期间，也可以利用cd命令从一个目录转移到其他任何目录中（只要访问权限允许）。例如，如果想要进入/usr/bin目录，可以输入下列命令：

```
$ cd /usr/bin
$
```

尽管目录文件通常是由系统维护的，用户不能直接写目录文件，但任何用户进程均可利用下列与目录操作有关的系统调用访问目录，自由地读取目录文件，只要具有相应的访问权限即可。

- `mkdir()`: 创建一个新目录
- `rmdir()`: 删除一个空目录
- `getdents()`: 获取目录文件的内容。

4.3.3 特殊文件

特殊文件是Linux系统中最具特色的特性之一。特殊文件也称设备文件。为了便于用户访问外部设备而不必涉及各种I/O设备的具体操作细节，Linux系统利用特殊文件作为用户与I/O设备之间的接口，使用户能够像读写普通文件一样访问外部设备。每个特殊文件均对应一个I/O设备，由I/O设备驱动程序实现用户与设备之间的数据通信。因此，用户可以像处理普通文件一样，通过打开、读写特殊文件等操作，实现访问外部设备的I/O处理要求。

特殊文件不包含任何数据。相反，特殊文件只是提供一种机制，在文件系统中建立一个物理设备与文件名之间的映射。系统支持的每个设备，包括内存，都与至少一个特殊文件相关联。特殊文件是利用下列`mknod`命令或系统调用创建的，而且必须提供相应的驱动程序，并集成到系统内核中。否则，即使创建了特殊文件，也无法访问相应的设备。

```
mknod special type [major minor]
```

其中，`special`为特殊文件名，如`/dev/console`（即系统控制台）。`major`为主设备号，表示按设备类型组织的设备驱动程序指针数组的索引。`minor`为次设备号，表示同类设备中的某个子设备，可以用做调用相应驱动程序的参数。`type`为特殊文件的类型，合法的特殊文件类型如下。

- `c`: 字符特殊文件;
- `b`: 块特殊文件;
- `p`: 管道文件。

例如，为了创建一个管道文件，可以使用下列命令：

```
$ mknod mypipe p
$ ls -l mypipe
prw-r--r-- 1 gqxing gqxing 0 2009-11-12 17:52 mypipe
$
```

当提交一个读写特殊文件的请求时，系统将会直接调用相应的设备驱动程序。而驱动程序则负责在控制进程与相关的物理设备之间传输数据。例如，假定存在下列两个命令（其中的`/dev/pts/0`是用户当前所用终端的设备名）：

```
$ cp /etc/passwd /tmp/garbage
$ cp /etc/passwd /dev/pts/0
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
.....
$
```

第一个命令只是把/etc/passwd文件的内容复制到/tmp/garbage文件中，创建或复制一个相应的文件。第二个命令则意味着把/etc/passwd文件复制到/dev/pts/0文件中，而后者实际上是一个与用户终端相关联的特殊文件。因此，/etc/passwd文件的内容将会显示在用户终端的屏幕上。

同UNIX系统一样，Linux系统也支持两种特殊文件：即字符特殊文件和块特殊文件。除了一个目录项和一个信息节点之外，设备特殊文件并不占用文件系统的任何空间，而这个特殊文件只是一个设备驱动程序的访问点。

Linux系统中的每一个设备或者是块设备，或者是字符设备，二者必居其一。通常，块设备主要用于存储数据，字符设备主要用于传输数据。例如，磁盘和CD/DVD等均属于块设备，连接到串口和并口的设备均属于字符设备。

设备除有块设备与字符设备等类型之分，还有主次设备号之分。主设备号主要用于区分不同的设备，次设备号主要用于区别同类设备的特定设备或分区。例如，假定磁盘设备的主设备号为8，次设备号1、2、……分别表示磁盘设备的第一个、第二个、……磁盘分区。

1. 块特殊文件

块特殊文件与采用数据块组织结构和处理方式的设备（如磁盘）相关联。所谓数据块组织结构的设备实际上指的是能够以固定长度的数据块传输数据，也能够随机访问其中任何数据块的存储设备，而且，访问每个数据块的时间差别也不大。

磁盘就是典型的数据块组织结构的设备。在Linux系统中，磁盘与系统内存之间通常是以数据块的方式传输数据的，以数据块为单位读写数据。借助文件系统，可以在磁盘的任何位置读写任意字节数量的数据。

在Linux系统中，一个典型的数据块是由512、1024或4096个字符组成的。因为文件系统通常都是驻留在块设备中的，故许多文件系统的维护命令均要求使用块特殊文件名作为文件参数。

例如，下列设备文件名就是一个对应于数据块组织结构的磁盘特殊文件（其中第一个字段的起始字符为b）：

```
$ ls -l /dev/sda2
brw-rw---- 1 root disk 8, 2 2009-11-19 14:32 /dev/sda2
$
```

当通过块特殊文件接口传输数据时，系统通常会在其内存（或高速缓冲区）中缓存数据。在指定的时间间隔内，系统将会把内存中的数据写入外部存储设备，从而更新存储设备中的数据。在交互访问或应用程序中，用户可以利用sync命令或fsync()系统调用，强制系统把内存中的数据写入存储设备，实现存储设备与内存之间的数据同步。

采用这种设计方式的一个问题是，如果在内存与存储设备之间进行数据同步之前，系统突然瘫痪，数据的完整性将会遭到破坏。因此，如果Linux系统出现故障，内存中尚未写到存储设备上的最新数据很可能会丢失。

更重要的是，内存中存有文件系统超级块中的最新数据，如果之前没有及时同步，文件系统（内存超级块与磁盘超级块之间）将会处于数据不一致的状态。如果再严重一点，文件系统有可能无法修复。

2. 字符特殊文件

任何非数据块组织结构的设备均为字符设备，而字符特殊文件就是与非数据块组织结构的任何设备相关联的特殊文件。与数据块组织结构的设备相反，字符设备无法使用定长的数据块，

也不能随机访问，其底层的I/O接口一次只能处理一个字符，这与块设备I/O接口一次能够处理一个完整的数据块的情况全然不同。

控制台终端、打印机、流式磁带机和其他串行设备均属于字符设备，因而具有相应的字符特殊文件名。当然，为了用于其他目的，数据块组织结构的设备（如磁盘）也具有相应的字符特殊文件名。

通过字符特殊文件接口传输的数据将会及设备驱动程序与控制进程之间直接传递，中间并不经过系统的数据缓冲区。因此，这种形式的设备接口经常称做原始接口。例如，下列设备文件名就是一个字符特殊文件（其中第一个字段的起始字符为c）：

```
$ ls -l /dev/console
crw----- 1 root root 5, 1 2009-11-19 14:33 /dev/console
$
```

3. 定义特殊文件

根据前述说明，每个设备都对应于一个特殊文件。通过读写特殊文件，可以实现对设备的I/O处理要求。实际上，这些特殊文件只是一个接口，设备的I/O处理都是由设备的驱动程序完成的。因此，每当在系统配置中增加一个新的设备时，都需要完成下列任务。

- （1）获取或编写相应设备的驱动程序；
- （2）更新系统配置表，描述新加的设备以及相应的设备驱动程序；
- （3）重新生成新的Linux内核，使其包含新的设备驱动程序；
- （4）利用mknod命令，创建新增设备的特殊文件。

4. 特殊文件实例——4个字符特殊文件

利用同一驱动程序模块与不同的次设备号，可以同时支持多个不同的字符特殊文件。例如，Linux系统通常均支持mem、kmem、null和null等4个比较有用的字符特殊文件，作为用户与系统内存之间的设备接口。例如，在安装Linux操作系统时，至少会自动创建下列3个（或4个）字符特殊文件：

```
$ cd /dev
$ ls -l mem null zero
crw-r----- 1 root kmem 1, 1 2009-11-19 14:32 mem
crw-rw-rw- 1 root root 1, 3 2009-11-19 14:32 null
crw-rw-rw- 1 root root 1, 5 2009-11-19 14:32 zero
$
```

mem、kmem、null和null等4个特殊文件的作用简述如下。

- /dev/null是一个数据回收站或漏斗文件，也是一个无底洞，只要写入这个文件，数据就会消失。这个文件经常用于过滤程序的输出。注意，当读取这个文件时，其功能等同于读取/dev/zero文件，用于返回指定数量的数值0。
- /dev/zero可以提供任意数量的数值0。例如，当程序从这个文件中读取256个字节时，将会收到256个0。如果写入这个文件，数据将会消失，其作用等同于/dev/null。
- /dev/mem提供计算机的物理内存接口，是Linux系统的内存映像。读取/dev/mem文件的字节地址将被解释为物理内存地址。如果从头开始（偏移值为0）读取这个文件，将会读取物理内存中从第一个字节开始的数据。

- /dev/kmem提供系统内核的虚拟内存接口。如果从头开始（偏移值为0）读取这个文件，将会读取系统内核从第一个字节开始的数据。如果没有这个接口，开发人员必须知道系统内核代码及其数据在物理内存中的起始位置。许多系统程序就是利用这个特殊文件访问系统内核，获取各种系统变量的内核地址，维护系统数据的。如果文件不存在，可以使用下列命令创建这个特殊文件：

```
# mknod -m 640 /dev/kmem c 1 2
# chown root:kmem /dev/kmem
```

在上述4个文件中，/dev/null最为常用。当我们不关心程序的标准输出，或标准错误输出，只是关注程序的出口状态时，经常会使用下列命令形式：

```
command > /dev/null
command > /dev/null 2>&1
```

4.3.4 链接文件

Linux系统提供一种机制，使之能够采用不同的文件名引用同一数据或程序，也即把同一数据或程序赋予不同的文件名。这种文件称做链接文件，也称硬链接文件。当调用不同目录中的同名程序时，或以不同的名字调用同一个程序时，执行的是同一程序副本。

采用链接文件的好处是，只需在文件系统中保存一份数据或程序副本，多个文件名均指向同一数据内容，更新任何一个文件，即可反映到其他文件中。既能节省磁盘存储空间，又可保证数据的一致性以及文件存储位置的灵活性。其中的一个例子就是罗马、圣马力诺和梵蒂冈的时区定义文件。

作为单独的文件，这3个时区定义文件位于同一文件系统的同一目录位置，只不过与之对应的文件数据（存储在磁盘中的数据块）只有一个副本。通过以下两个命令，即可验证上述说法：

```
$ cd /usr/share/zoneinfo/Europe
$ ls -il Rome
25340 -rw-r--r-- 3 root root 2652 2009-11-12 21:22 Rome
$ ls -il | grep ^25340
25340 -rw-r--r-- 3 root root 2652 2009-11-12 21:22 Rome
25340 -rw-r--r-- 3 root root 2652 2009-11-12 21:22 San_Marino
25340 -rw-r--r-- 3 root root 2652 2009-11-12 21:22 Vatican
$
```

上述3个文件具有相同的信息节点号，且其链接计数（第3列）为3，说明这3个文件是采用硬链接方式链接在一起的，因而拥有同一数据副本，但具有不同的文件名。注意，除了目录之外，非链接文件（包括符号链接文件）的链接计数只能为1。在Linux系统中，信息节点与数据是一一对应的。如果文件的信息节点号相同，其引用的就是同一个信息节点，因而拥有同一数据副本，如图4-3所示。

若想创建硬链接文件，可以使用ln命令或link()系统调用实现。创建硬链接文件时，文件引用的是同一信息节点号和文件数据，只是在同一目录或不同目录中增加一个新的文件名而已。注意，硬链接文件的唯一局限是链接的两个文件必须位于同一个物理文件系统中。

在Linux系统中，对于同一个数据内容，出于应用的需要，经常会赋予其不同的文件名。例如，/etc/init.d目录中的Shell脚本文件主要用于启动或关闭系统守护进程。同一个文件又可用

于不同的运行级，因而需要放置到不同的目录中，如放置到/etc/rc0.d或/etc/rc2.d等目录中。

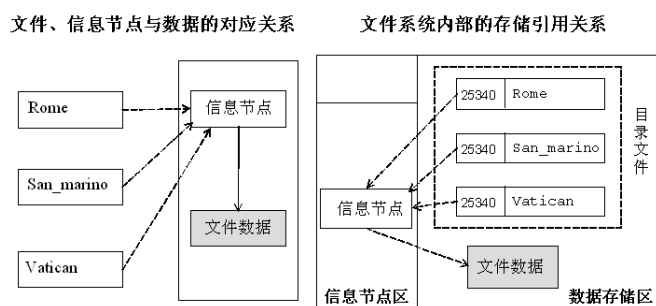


图4-3 硬链接文件的数据引用关系

例如，在Ubuntu 8.04 Linux系统中，安装Samba服务器软件包时仅在/etc/init.d目录中增加了一个samba脚本，没有在rcN.d目录中增加必要的链接文件。为了把文件的副本分别存放到/etc/rc2.d和/etc/rc0.d两个目录中，以便达到自动启动与关闭Samba服务器的目的，可以使用ln命令，创建两个链接文件。

```
$ cd /etc/init.d
$ sudo ln samba ../rc2.d/S90samba
$ sudo ln samba ../rc0.d/K10samba
$ ls -l samba
-rwxr-xr-x 3 root root 2932 2008-09-10 20:10 samba
$ ls -l ../rc2.d/S90samba
-rwxr-xr-x 3 root root 2932 2008-09-10 20:10 ../rc2.d/S90samba
$ ls -l ../rc0.d/K10samba
-rwxr-xr-x 3 root root 2932 2008-09-10 20:10 ../rc0.d/K10samba
$
```

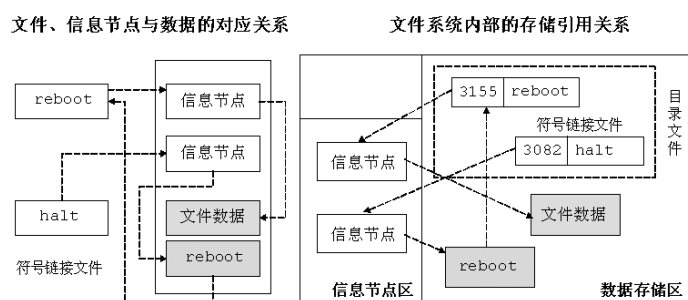
如此一来，就建立了三个文件，三个文件共享同一个数据副本。也就是说，三个文件名均指向同一数据内容。

4.3.5 符号链接文件

类似于UNIX系统，Linux系统也提供一种新的文件链接机制，以便用户能够跨越不同的物理文件系统建立链接文件，这种新的链接文件称做符号链接文件。为了创建符号链接文件，可以使用“ln -s”命令或symlink()系统调用实现。

与硬链接文件的实现方式不同，符号链接文件的本身也是一种数据文件，而且是一个单独的文件，只不过文件中的内容并非真正的数据，而是一个指向目的文件（或目录）的路径名而已。由此可见，利用符号链接机制，将会创建新的文件名和新的信息节点。而且，符号链接文件也会拥有自己的数据块，其中包含的就是其引用的目的文件（或目录）的完整路径名。例如，常用的reboot命令是一个基本文件，halt（包括poweroff）等命令只是其符号链接文件，如图4-4所示。

当引用符号链接文件时，系统首先需要打开符号链接文件本身，然后再根据文件中的内容，打开其指向的文件。因此，当操作一个符号链接文件时，系统将会按照符号链接文件中的路径名进行二次检索，直至找到实际的文件。此外，符号链接文件与其链接的目的文件不必位于同一物理文件系统中，其引用的文件甚至可以不存在。符号链接文件可用于引用任何文件，尤其是目录，但硬链接不能链接目录，以免出现目录树引用的死循环。



采用符号链接文件的最大好处是能够确保目录文件结构的兼容性。当Linux系统的目录结构与文件组织需要调整时，为了保持文件系统的兼容性（这也是产生符号链接文件的主要原因之一），可以采用符号链接文件的形式，继续保持原有的目录结构或文件位置。例如，Ubuntu Linux系统把/etc/motd文件迁移至/var/run/motd，但仍然保持/etc/motd文件的位置，只是把这个文件变成了符号链接文件，如下所示：

```
$ ls -ld /etc/motd
lrwxrwxrwx 1 root root 13 2009-11-13 02:38 /etc/motd -> /var/run/motd
$
```

利用符号链接文件，还可以照顾用户以往的上机习惯，把之前常用的命令名链接到新增的命令，实现命令名字的借用或间接引用。例如，halt、poweroff与reboot命令引用的实际上是同一个reboot程序，vi与vim命令引用的是同一vim程序。

在符号链接文件中，文件大小字段中的数值恰好等于符号链接文件引用的目的文件的路径名的长度（观察第5列与“->”符号后的路径名长度）：

```
$ cd /etc/rc2.d
$ ls -l S*
.....
lrwxrwxrwx 1 root root 18 2009-11-13 02:38 S99ondemand -> ../init.d/ondemand
lrwxrwxrwx 1 root root 18 2009-11-13 02:38 S99rc.local -> ../init.d/rc.local
$
```

Linux中也存在3个与符号链接文件有关的系统调用，其目的是访问符号链接本身及其引用的目的文件或目录的相关信息。这些系统调用如下。

- readlink(): 用于读取符号链接文件引用的目的文件或目录的路径名。这些信息存储在符号链接文件的数据块中。
- lstat(): 其功能类似于stat系统调用，除用于获取目的文件的属性信息外，还可用于获取符号链接文件本身的有关信息。
- lchown(): 类似于chown系统调用，既可用于修改目的文件的属主属性，也可用于修改符号链接文件本身的属主属性。

如果想把上述的samba文件改为符号链接方式，可以利用ln命令的“-s”选项，创建下列两个符号链接文件：

```
$ cd /etc/init.d
$ sudo ln -s samba ../rc2.d/S90samba
```

```

$ sudo ln -s samba ../rc0.d/K10samba
$ ls -l samba
-rwxr-xr-x 1 root root 2932 2008-09-10 20:10 samba
$ ls -l ../rc2.d/S90samba
lrwxrwxrwx 1 root root 5 2008-09-10 20:10 ../rc2.d/S90samba -> samba
$ ls -l ../rc0.d/K10samba
lrwxrwxrwx 1 root root 5 2008-09-10 20:10 ../rc0.d/K10samba -> samba
$

```

在实际应用中, 链接文件具有重要的用途。例如, 在构建互为备份的双机系统中, 通常采用两个系统共享同一磁盘阵列的形式, 实现数据共享。在这种实现方式中, 有的HA软件要求比较高, 如要求磁盘阵列的设备名必须一致。而由于种种原因, 这一限制或要求可能很难达到。为了解决这个问题, 便可以利用ln命令建立链接或符号链接文件。

4.3.6 管道文件

Linux系统存在两种管道: 普通管道(简称管道)和管道文件。普通管道是一个可用文件描述符标识和存取的数据缓冲区, 管道内的数据按先进先出的方式处理。普通管道通常是在程序中利用pipe(2)系统调用建立的。当程序执行结束后, 管道也就自动消失了。

管道文件与普通管道的功能基本相同, 只是其创建的方式不同。管道文件是通过下列mknod命令或mknod()系统调用创建的, 而且它作为一个特殊文件, 存在于文件系统中, 故管道文件也称命名的管道(named pipe):

```
# mknod pipefile p
```

普通管道是一种进程之间的通信机制, 而管道文件则是一种特殊文件。管道文件用于缓存接收到的数据, 同时供从管道文件中读取数据的进程按照先进先出(FIFO)的方式读取数据。尽管管道文件具有文件名和信息节点, 但这个特殊文件并不拥有任何数据。

在下面的例子中, usplash_fifo就是一个管道文件(第一列文件属性中的第一个字符p即管道文件的标志)。

```

$ ls -l /dev/.initramfs/usplash_fifo
prw-r----- 1 root root 0 2009-11-19 14:32 /dev/.initramfs/usplash_fifo
$

```

4.4 文件的安全保护机制

Linux是一个多用户、多任务的操作系统, 为了保证系统和信息的安全, 防止用户间未经许可擅自访问他人的文件等, Linux系统从信息的基本组织单位——文件出发, 把用户分为三类: 文件属主、同组用户和其他用户(也即所有用户)。

对于任何一个文件, 可以针对三类用户分别赋予一定的访问权限。这些访问权限分为读、写和执行。表4-2描述了文件的三种基本访问权限及其意义。

通常, 任何用户都可能会允许其他用户读自己的文件, 但大多不会允许别人对文件做任何修改。对于普通的程序文件, 也可能会允许任何人都能执行。Linux系统的文件目录安全机制使用户能够控制文件和目录的访问权限。

表4-2 文件的三种基本访问权限

访问权限	简单说明
r（读）	如果文件具有读许可，相应的用户可以读文件，如显示文件内容，复制文件等，但不能修改文件。如果允许用户进入某个目录（cd命令），列举目录下的文件，至少应赋予用户“读”目录的访问权限
w（写）	如果文件具有写许可，相应的用户可以读、写文件，包括显示文件内容，复制、修改、移动和删除文件等。对于目录而言，如果允许用户创建新文件和删除文件，必须赋予用户“写”目录的访问权限
x（执行）	如果具有执行许可，相应的用户可以运行文件（如程序文件）。对于目录而言，如果允许用户访问其中的任何子目录，必须赋予用户“执行”目录的访问权限

4.4.1 显示文件的访问权限

为了了解文件的访问权限及其属性，可以使用带有“-l”选项的ls命令，按文件名的字符顺序以长格式显示每一个文件的各种属性信息，如图4-5所示。

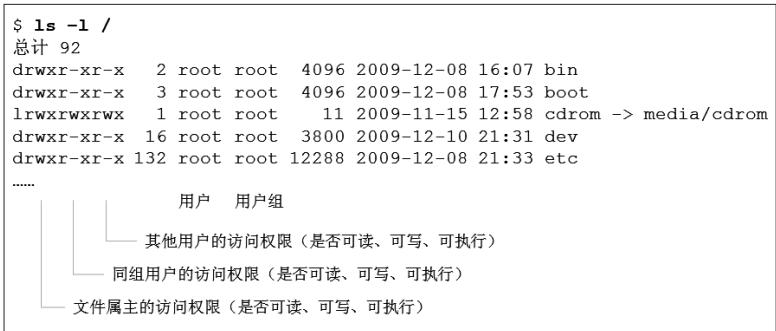


图4-5 显示文件目录的访问权限

如前所述，ls命令输出的第一列中的第一个字符表示文件的类型。后面9个字符表示文件或目录的访问权限。这些访问权限可分为三组，分别对应三组用户：文件属主、同组用户和其他用户。前三个字符为一组，表示文件属主对此文件的访问权限；中间三个字符为一组，表示与文件属主同组的用户对此文件的访问权限；最后三个字符为一组，表示其他所有用户对此文件的访问权限。每一组的三个字符依次为r、w和x，分别表示读、写和执行。

如果r、w或x存在，则表示相应的用户分别具有读、写或执行的访问权限。下面，我们以/etc目录和/etc/profile文件为例予以说明：

```
$ ls -ld /etc
drwxr-xr-x 131 root root 12288 2009-11-19 14:33 /etc
$ ls -l /etc/profile
-rw-r--r-- 1 root root 497 2009-10-29 05:49 /etc/profile
$
```

在上述例子中，/etc目录的访问权限是“drwxr-xr-x”，表示每个人都能读和执行这个目录，但只有目录的属主root才能写这个目录。/etc/profile文件的访问权限是“-rw-r--r--”，表示文件属主root能够读写这个文件，同组用户和其他用户只能读这个文件，但任何人（包括文件属主）都不能执行这个文件。

4.4.2 修改文件的访问权限

为了修改文件或目录的访问权限，可以使用`chmod`命令。修改文件或目录访问权限的前提是，用户必须是文件或目录的属主，具有修改文件访问权限的权力，或者是超级用户。`chmod`命令的语法格式如下：

```
chmod permissions dir-or-file
```

其中，`permissions`表示新的访问权限，`dir-or-file`是准备修改其访问权限的文件或目录。

在设置文件或目录的访问权限时，可以对现有的访问权限进行增减性的调整，也可以直接设置。下面简单说明怎样在现有文件访问权限的基础上进行修改。

(1) 使用一个或多个字符表示用户的类型：

- `u` (表示文件属主)；
- `g` (表示同组用户)；
- `o` (表示其他用户)；
- `a` (表示所有用户)。

(2) 使用加号“+”和减号“-”分别表示增加或撤销相应的访问权限。

(3) 使用一个或多个字符表示实际的访问权限：

- `r` (表示读)；
- `w` (表示写)；
- `x` (表示执行)。

在下面的`chmod`命令中，“`o+w`”表示增加其他用户写`script`目录的访问权限。

```
$ cd /home/gqxing
$ ls -ld script
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-08 09:10 script
$ chmod o+w script
$ ls -ld script
drwxr-xrwx 2 gqxing gqxing 4096 2009-11-08 09:10 script
$
```

从“`ls -l`”命令的输出结果可以看出，执行“`chmod o+w script`”命令之后，`script`目录相应于其他用户写目录的权限标志已由“-”变为“`w`”。

为了撤销其他用户读和执行同一目录的访问权限，可使用下列`chmod`命令：

```
$ ls -ld script
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-08 09:10 script
$ chmod o-rx script
$ ls -ld script
drwxr-x--- 2 gqxing gqxing 4096 2009-11-08 09:10 script
$
```

执行上述`chmod`命令之后，表示其他用户读和执行目录的权限标志已由“`r`”和“`x`”全部变为“-”。

通常，普通文件的默认访问权限不包括执行权限。因此，在新建了一个`settcp`脚本文件之后，为了确保文件属主能够执行脚本文件，可以使用下列`chmod`命令：

```
$ ls -l settcp
-rw-r--r-- 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$ chmod u+x settcp
$ ls -l settcp
-rwxr--r-- 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$
```

如果想增加或撤销所有用户的访问权限，可以使用chmod命令的“a+”或“a-”选项。例如，为了使所有的用户都能够执行settcp文件，可输入下列chmod命令：

```
$ ls -l settcp
-rw-r--r-- 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$ chmod a+x settcp
$ ls -l settcp
-rwxr-xr-x 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$
```

在上述输出信息中，访问权限字段（第一列后9个字符）中的三个“x”表示所有用户均可以执行settcp文件。

此外，还可以使用星号“*”通配符，同时修改多个文件的访问权限。例如，为了设置当前目录中所有文件的访问权限，使得除了文件属主能够写之外，其他所有用户均不能修改这些文件，可以使用下列chmod命令：

```
$ ls -l
总计 16
-rwxrwxrwx 1 gqxing gqxing 1608 2009-11-08 09:18 setftp
-rwxrwxrwx 1 gqxing gqxing 4589 2009-11-08 09:20 setnfs
-rwxrwxrwx 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$ chmod go-w *
$ ls -l
总计 16
-rwxr-xr-x 1 gqxing gqxing 1608 2009-11-08 09:18 setftp
-rwxr-xr-x 1 gqxing gqxing 4589 2009-11-08 09:20 setnfs
-rwxr-xr-x 1 gqxing gqxing 1670 2009-11-08 09:26 settcp
$
```

4.4.3 设置文件的访问权限

上一节讨论了怎样使用chmod命令在文件现有访问权限的基础上进行调整的情况。这种文件访问权限设置法称为相对权限设置法，这一节将要讨论的方法可以称做绝对权限设置法——使用chmod命令和数字代码，对文件的访问权限直接进行设置。chmod命令的语法格式如下：

```
chmod numcode dir-or-file
```

其中，numcode是一个数字代码，用于表示文件的访问权限。dir-or-file是文件或目录的名字。

表示文件访问权限的数字代码由三位数字组成，分别对应于文件属主、同组用户和其他用户。例如，下列chmod命令设置的访问权限表示：setftp文件的属主能够读、写和执行setftp文件，而其他用户只有读和执行文件的权限：

```

$ ls -l setftp
-rw-r--r-- 1 gqxing gqxing 1608 2009-11-08 09:18 setftp
$ chmod 755 setftp
$ ls -l setftp
-rwxr-xr-x 1 gqxing gqxing 1608 2009-11-08 09:18 setftp
$

```

表4-3解释了数字代码755表示的访问权限,以及755数字代码的由来。表中的后面3列每列对应一类用户。

表4-3 setftp文件访问权限的设定

访问权限	文件属主	同组用户	其他用户
读	100 (4)	100 (4)	100 (4)
写	010 (2)	000 (0)	000 (0)
执行	001 (1)	001 (1)	001 (1)
全部访问权限	7	5	5

表示访问权限的字符串“rwx”可以看做3个二进制数据,如果“r”存在,则表示相应数据位为1,写成二进制数就是“100”,也就是十进制的4。同样,“w”对应的二进制数是“010”,即十进制的2。“x”对应的二进制数是“001”,也即十进制的1。

为了设置读文件的访问权限,可在适当的行列交叉位置加一个4。为了设置写文件的访问权限,可在适当的行列交叉位置加一个2。同样,为了设置执行文件的访问权限,可在适当的行列交叉位置加一个1。把3列的3行数值分别加在一起,然后再把这3个数据组合起来,就是一个表示文件访问权限的完整数字代码。

例如,为了使所有的用户都能够读、写和执行setnfs文件,可以使用下列命令。其中,数字代码777是能够设置的最大文件访问权限:

```

$ ls -l setnfs
-rw-r--r-- 1 gqxing gqxing 4589 2009-11-08 09:20 setnfs
$ chmod 777 setnfs
$ ls -l setnfs
-rwxrwxrwx 1 gqxing gqxing 4589 2009-11-08 09:20 setnfs
$

```

表4-4解释了上述访问权限数字代码的意义及由来。

表4-4 设置setnfs文件访问权限的数字代码

访问权限	文件属主	同组用户	其他用户
读	4	4	4
写	2	2	2
执行	1	1	1
全部访问权限	7	7	7

类似于相对权限设置法，在采用绝对权限设置法时也可以使用星号“*”通配符，对选定的文件设置同一访问权限。例如，我们可以使用`chmod`命令，统一设置当前目录下所有文件的访问权限：

```
$ ls -l
总计 16
-rw-r--r-- 1 qqxing qqxing 1608 2009-11-08 09:18 setftp
-rw-r--r-- 1 qqxing qqxing 4589 2009-11-08 09:20 setnfs
-rw-r--r-- 1 qqxing qqxing 1670 2009-11-08 09:26 settcp
$ chmod 755 *
$ ls -l
总计 16
-rwxr-xr-x 1 qqxing qqxing 1608 2009-11-08 09:18 setftp
-rwxr-xr-x 1 qqxing qqxing 4589 2009-11-08 09:20 setnfs
-rwxr-xr-x 1 qqxing qqxing 1670 2009-11-08 09:26 settcp
$
```

上述例子中的两个`ls`命令解释了使用`chmod`命令前后文件访问权限的变化。

使用绝对权限设置的最大好处是，无需知道文件当前的访问权限，用户只需按照自己的意愿直接设置即可。

4.4.4 其他访问权限设置

1. 默认访问权限

在创建任何目录或文件时，系统通常都会赋予新建目录或文件一个访问权限。如果没有设置用户掩码（user mask），系统默认的目录访问权限为777（对应于二进制的111 111 111），文件访问权限为666（对应于二进制的110 110 110）。如果设置了用户掩码，目录和文件的实际访问权限是由系统访问权限与用户掩码经过逻辑异或运算后的结果。

用户掩码通常包含三组八进制的数字，其取值范围为0~7：

- 第一组数字用于设置用户自己的访问权限；
- 第二组数字用于设置同组用户的访问权限；
- 第三组数字用于设置其他用户的访问权限。

通常，系统设置的用户掩码为022（对应于二进制的000 010 010）。经过逻辑异或运算之后，可知新建目录的访问权限为755（对应于二进制的111 101 101），新建文件的访问权限为644（对应于二进制的110 100 100）。

例如，当创建一个新文件时，系统自动设置的访问权限如下：

```
-rw-r--r--
```

当创建一个新目录时，系统自动设置的访问权限如下：

```
drwxr-xr-x
```

用户掩码是由`umask`命令设置的（在系统提供的`/etc/profile`初始化文件中，`umask`命令通常是一个不可或缺的设置命令。在用户注册之后，通过执行`/etc/profile`文件，即可实现用户掩码的设置，从而达到设置目录和文件访问权限的目的）。`umask`命令的语法格式如下：

```
umask [-S] [nnn]
```


其中, `nnn`是一个3位的八进制数值, 表示用户掩码。注意, 这个用户掩码与`chmod`命令中的数字代码意义恰好相反。在`chmod`命令中, 八进制的数字代码777意味着任何人都具有读、写和执行文件的访问权限, 而在`umask`命令中, 数字代码777意味着任何人都没有访问权限, 包括文件属主本人也是如此。

为了查询系统当前的用户掩码设置, 可以直接输入`umask`命令, 示例如下:

```
$ umask
022
$
```

当需要使用`umask`命令设置文件的默认访问权限时, 需要确定究竟应采用什么用户掩码值。具体做法是, 从系统默认的文件访问权限666中减去文件的实际访问权限对应的八进制数值, 余数即可用于`umask`命令中的用户掩码。例如, 假定我们想把文件的访问权限设置为664 (`rw-rw-r--`), 而666与664的差为002, 故可把002用做`umask`命令的参数。如果运行“`umask 000`”命令, 意味着把文件的访问权限设置为666 (`rw-rw-rw-`)。

也可以把二进制的“000 000 000”看做文件访问权限的基础, 其中0表示允许, 1表示禁止。如果想屏蔽某一访问权限, 可把相应的二进制数值置1, 最后再转换成八进制的数字代码即可。例如, 如果用户期望自己能够读写新建的文件, 同组和其他用户只能读文件, 则其二进制数字代码应为“000 010 010”, 转换为八进制数字代码即为“022”。因此, 可以使用“`umask 022`”命令设置用户掩码, 从而设置文件的默认访问权限。

2. 设置特殊的访问权限

前面我们已经讨论了如何设置文件的访问权限, 任何用户均可用以保护自己的文件。还有一些特殊的文件访问权限, 可由超级用户进行设置。Linux系统中的许多管理文件, 只有超级用户才有权修改, 但普通用户有时也需要能够像超级用户一样做必要的修改。例如, 管理用户账号和密码的`/etc/passwd`和`/etc/shadow`文件, 普通用户是不能修改的。但用户经常需要修改自己的密码, 而普通用户又不能修改这些文件, 总不能每次都经过系统管理员。

为此, Linux系统也采用了实际用户ID (实际用户组ID) 和有效用户ID (有效用户组ID) 的概念。通过设置有效用户ID, 使用户能够临时地借用超级用户身份执行某些特权命令。例如, 任何用户在使用`passwd`命令修改密码期间, 其有效用户ID暂时变为超级用户ID, 因而可以访问`shadow`等文件, 修改自己的密码。一旦退出`passwd`命令, 用户的有效ID又恢复原状。

为了设置某一命令文件的有效用户ID或有效用户组ID, 超级用户可以分别使用下列命令实现:

```
# chmod u+s filename
# chmod g+s filename
```

在设置了有效用户ID之后, 必须先由超级用户执行一次相应的命令, 才能使上述设置发挥作用。当普通用户执行相应的命令时, 其有效用户ID才能变为超级用户的有效用户ID。注意, 有效用户ID仅在执行过程中才能生效, 因此, 只有对命令或程序文件设置有效用户ID才有实际的意义。

第5章 文件与目录操作

文件和目录操作是每个系统都必不可少的基本功能。Linux系统提供大量的文件与目录处理的命令和工具，用于创建、显示、移动、删除、复制和维护文件与目录。本章主要介绍Ubuntu Linux系统中的各种文件与目录操作命令，详细说明怎样创建、显示、移动、删除、复制和维护文件与目录。在本章的后面，我们还将介绍若干有用的实用程序，如find、diff、grep和sort等。

5.1 创建文件

尽管没有专门的命令用于创建新文件，但创建新文件的方法有许多种。其中第一种方法是利用touch命令创建一个新的空文件。touch命令的本意是以当前（或指定）的时间更新给定文件的访问和修改时间。如果指定的文件不存在，则创建一个新的空文件，例如：

```
$ touch emptyfile
$
```

而第二种方法相对更简单，即借用重定向符号“>”，创建一个新的空文件：

```
$ > emptyfile
$
```

第三种方法是采用echo命令与重定向符号“>”创建一个新文件。echo命令的本意是显示提示或说明信息。这里借用echo命令，把其标准输出重定向到一个文件中，从而创建一个含有一行数据的新文件：

```
$ echo "Only one line in file" > newfile
$
```

第四种方法是采用cat命令与重定向符号“>”创建一个新文件。cat命令主要用于显示文件的内容。这里借用cat命令读取标准输入的特点，直接从键盘上输入数据，从而创建一个拥有多行数据的新文件。其中，“Ctrl-D”是Linux系统中的文件结束符。例如：

```
$ cat > myfile
The text I am typing will be stored in myfile.
Press Enter key at the end of each line.
When finished, hold down the Ctrl key and press D.
Ctrl-D (不显示)
$
```

此外，还可以使用vim等文本编辑器创建和编辑文件，详见第6章“Vim编辑器”。实际上，创建文件的方法还有很多，熟悉了Linux系统之后，读者还会找出其他更多的方法。

5.2 显示文件列表

5.2.1 使用ls命令列出文件

了解系统中都有什么文件，了解自己当前目录下都有哪些文件，也许是用户最常见的操作了。为此，可以使用ls命令。ls命令的语法格式简写如下：

```
ls [-abdhiklrRs] [dir-or-file]
```

ls命令具有很多选项，表5-1给出了部分常用的选项。

表5-1 ls命令的部分常用选项

选项	GNU选项	简单说明
-a	--all	列出指定（或当前）目录下的所有文件，包括以句点“.”作为起始字符的隐藏文件
-b	--escape	当文件名包含不可打印的特殊字符时，以八进制数字形式列出文件的名字
-d	--directory	在使用ls命令列举文件时，如果指定的参数是一个目录，仅列出目录的名字，而不是列出目录下的文件。“-d”选项经常与“-l”选项一起使用，以便了解目录的属性信息
-h	--human-readable	以KB、MB和GB的形式显示文件的大小（这个选项应与“-l”或“-s”等选项一同使用）
-i	--inode	对于每一个文件，在第一列列出其信息节点号
-k	--block-size=1 K	以KB为单位给出文件的大小
-l	--format=long	以每行一个文件的长格式列出文件的类型、访问权限、链接数、用户属主、用户组、文件大小、最后修改时间和文件名等信息。如果是特殊文件，文件大小字段分为两个数字字段，分别表示设备的主、次设备号。如果是一个符号链接文件，则以“filename→被引用文件的路径名”的形式给出文件名字段
-r	--reverse	以文件名反向字符排序的顺序显示文件列表
-R	--recursive	递归地列出指定（或当前）目录及其子目录下的所有文件
-s	--size	显示分配给文件的数据块（1024字节）的数量，也即文件占用的数据块数量，而非文件的实际大小。因此，文件大小相近的文件，“-s”选项给出的结果完全可能是一样的

例如，为了列出当前目录下的文件，最简单的ls命令形式如下：

```
$ ls
emptyfile  myfile  newfile
$
```

通常，ls命令将会按照文件名的字符顺序列出当前目录下的所有文件。上述例子中的ls命令未加任何选项，也没有明确地指定目录或文件参数，则默认为当前目录。输出结果说明当前目录下只有三个刚才创建的文件。如果了解其他某个特定目录下都有什么文件，可在ls命令后面明确地指定目录名，例如：

```
$ ls /
bin      dev      initrd.img media proc  selinux  tmp      vmlinuz
boot     etc      lib       mnt   root  srv       usr
cdrom    home    lost+found opt    sbin  sys       var
$
```

上述ls命令按照文件名的字符顺序列出了根目录下的所有文件。读者可能已经发现，使用ls命令而不加任何选项，系统仅仅列出文件的名字，至于这些文件究竟是目录、普通文件、特殊文件、管道文件，还是链接文件，则不得而知。为了能够了解更多的信息，可使用“-l”等选项，例如：

```
$ ls -l
总计 8
-rw-r--r-- 1 gqxing gqxing  0 2009-11-10 18:31 emptyfile
-rw-r--r-- 1 gqxing gqxing 139 2009-11-10 18:36 myfile
-rw-r--r-- 1 gqxing gqxing  22 2009-11-10 18:33 newfile
$
```

使用“-l”选项，ls命令将会按照一行8列（注意，如采用纯英文语言环境，应为9列）的形式，逐行显示每一个文件的属性。其中第一列包含10个字符，第一个字符表示文件的类型。常见的文件类型字符概述如下：

- -：表示相应的文件是一个普通文件；
- d：表示相应的文件是一个目录；
- l：表示相应的文件是一个符号链接文件；
- b：表示相应的文件是一个块特殊文件；
- c：表示相应的文件是一个字符特殊文件；
- p：表示相应的文件是一个管道（FIFO）文件；
- s：表示相应的文件是一个套接字文件。

其余9个字符可分为三组，分别表示文件属主、同组用户和其他用户对相应文件的访问权限，详见第4章的讨论。第二列是一个数字，表示文件的链接数。如果此数字大于1，说明同一文件数据存在多个文件名字。第三列为用户名，表示文件的属主，说明文件归谁拥有。第四列是文件属主所在用户组的名字。第五列是文件以字节为单位的大小。第六列至第八列是文件最后一次修改的日期和时间（注意，中文语言环境与英文语言环境的输出结果稍有不同：前者为8列，后者为9列）。最后一列才是文件的名字，如图5-1所示。

另外，为了照顾DOS/Windows用户的习惯，Linux系统也提供一个dir命令，只不过除了命令名不同之外，其命令选项及输出形式与ls命令完全相同，但与DOS/Windows系统中的情况并不相同。可以说，dir命令只是ls命令的一个替代品。例如：

```
$ dir
emptyfile myfile newfile
$ dir -l
总计 8
-rw-r--r-- 1 gqxing gqxing  0 2009-11-10 18:31 emptyfile
-rw-r--r-- 1 gqxing gqxing 139 2009-11-10 18:36 myfile
-rw-r--r-- 1 gqxing gqxing  22 2009-11-10 18:33 newfile
$
```

drwxr-xr-x	131	root	root	12288	Nov 19 12:30	/etc
-rw-r--r--	1	root	root	497	Oct 29 05:49	/etc/profile

链接数 用户组 修改日期 时间 文件名
 文件属主 文件大小（以字节计）

— 其他用户的访问权限（是否可读、可写、可执行）
 — 同组用户的访问权限（是否可读、可写、可执行）
 — 文件属主的访问权限（是否可读、可写、可执行）
 — 文件的类型（“-”表示普通文件，“d”表示目录）

brw-rw----	1	root	disk 8, 2	Nov 19 12:26	sda2
crw-----	1	root	root 5, 1	Nov 19 12:30	console

链接数 用户组 主设备号 修改日期 时间 文件名
 文件属主 次设备号

— 其他用户的访问权限（是否可读、可写、可执行）
 — 同组用户的访问权限（是否可读、可写、可执行）
 — 文件属主的访问权限（是否可读、可写、可执行）
 — 文件的类型（“b”表示块特殊文件，“c”表示字符特殊文件）

图5-1 文件属性

5.2.2 利用通配符显示文件

在第3章中，我们已经介绍了通配符的作用及其与文件名之间的关系。当需要处理一组具有某一共同属性的文件时，可以利用通配符实行模式匹配，生成一个具有同一属性的文件列表。例如，在Linux系统中，C程序大多以“.c”作为文件名后缀。为了显示当前目录下的所有C程序，可以使用下列命令：

```
$ ls -l *.c
-rw-r--r-- 1 gqxing gqxing 16188 2009-11-18 14:03 atmcom.c
-rw-r--r-- 1 gqxing gqxing 28648 2009-11-18 14:04 atmmon.c
-rw-r--r-- 1 gqxing gqxing 76360 2009-11-18 14:05 handler.c
-rw-r--r-- 1 gqxing gqxing 28080 2009-11-18 14:06 listener.c
$
```

在上述命令中，星号“*”就是一个通配符，能够匹配任何字符或字符串。实际上，可以利用的通配符还有问号“?”（用以匹配任何一个字符），以及字符集“[...]”（用以匹配字符集中的任何一个字符）等。

如前所述，如果不提供文件名参数，ls命令将会列出当前目录下的所有文件，而星号“*”通配符又能够匹配任何文件，如果使用下列两个ls命令，其输出结果会有什么不同呢？

```
ls -l
ls -l *
```

上述的第一个ls命令中没有指定任何目录或文件参数，因此，系统将会把当前目录作为默认的参数，列出当前目录下的所有文件。第二个ls命令使用了星号“*”通配符作为目录或文件参数，而此处的星号“*”通配符则表示当前目录下的所有文件。因此，针对第二个ls命令，Shell将会使用当前目录下的每一个文件作为参数提供给ls命令。如果当前目录下只有普通文件，则上述两个命令将会产生同样的输出结果。但是，如果当前目录下含有任何子目录，第一个ls命令只是列出子目录的名字，而第二个ls命令不仅会列出子目录的名字，还将列出子目录下的所有文件。示例如下：

```
$ cd /etc/power
$ ls -l
总计 8
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 event.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 scripts.d
$ ls -l *
event.d:
总计 4
-rwxr-xr-x 1 root root 811 2009-10-07 07:02 laptop-mode
scripts.d:
总计 4
-rwxr-xr-x 1 root root 874 2009-10-07 07:02 laptop-mode
$
```

在使用星号“*”通配符的情况下，为了避免列出子目录中的文件，可以使用“-d”选项。示例如下：

```
$ ls -ld *
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 event.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 scripts.d
$
```

5.2.3 显示隐藏文件

在Linux系统中，如果文件名的第一个字符为句点“.”，如“profile”、“bash_history”和“bashrc”，这样的文件称做隐藏文件。通常，ls命令不会列出隐藏文件。但是，使用ls命令的“-a”选项，则可以列出这种隐藏的文件，例如：

```
$ ls -la
总计 2112
drwxr-xr-x 51 gqxing gqxing 4096 2009-11-20 13:02 .
drwxr-xr-x 4 root root 4096 2009-11-15 16:12 ..
drwx----- 2 gqxing gqxing 4096 2009-11-18 01:53 .aptitude
-rw----- 1 gqxing gqxing 702 2009-11-20 15:09 .bash_history
-rw-r--r-- 1 gqxing gqxing 220 2009-11-15 16:12 .bash_logout
-rw-r--r-- 1 gqxing gqxing 3180 2009-11-15 16:12 .bashrc
.....
$
```

在UNIX系统中，隐藏文件通常位于文件列表的最前面，而在Linux系统中，隐藏文件将会在不考虑第一个句点“.”字符的情况下同普通文件一样参与排序。文件名仅有一个句点字符“.”的文件表示当前目录，文件名为双句点“..”的文件表示父目录。除此之外，其他隐藏文件通常均用于设置用户的工作环境。

此外，当由于某种原因，在命名文件时误输入了不可见的特殊字符时，ls命令可能无法正确地列出文件的名称。而在使用文件处理命令操作这样的文件时，可能会显示“No such file or directory”等错误信息。例如，见到下面的信息，用户可能会感到莫名其妙，同一目录下怎么会有两个同名的文件呢？这是完全不符合Linux文件系统的规定的。

```
$ ls -l
总计 16
-rw-r--r-- 1 qqxing qqxing 1268 2009-11-10 19:00 fi?le3
-rw-r--r-- 1 qqxing qqxing 1998 2009-11-10 18:57 fi?le3
-rw-r--r-- 1 qqxing qqxing 1663 2009-11-10 18:56 file1
-rw-r--r-- 1 qqxing qqxing 2556 2009-11-10 18:57 file2
$ ls -l fi?le3
-rw-r--r-- 1 qqxing qqxing 1268 2009-11-10 19:00 fi?le3
-rw-r--r-- 1 qqxing qqxing 1998 2009-11-10 18:57 fi?le3
$
```

上述两个文件的大小并不相同，明显不是同一个文件。原来，当文件名中包含控制字符等不可打印的特殊字符时，Linux系统将会采用问号“?”替代任何不可打印的特殊字符。为此，可以使用ls命令的“-b”选项，以八进制数字的形式列出文件名中不可见的特殊字符：

```
$ ls -lb
总计 16
-rw-r--r-- 1 qqxing qqxing 1268 2009-11-10 19:00 fi\002le3
-rw-r--r-- 1 qqxing qqxing 1998 2009-11-10 18:57 fi\024le3
-rw-r--r-- 1 qqxing qqxing 1663 2009-11-10 18:56 file1
-rw-r--r-- 1 qqxing qqxing 2556 2009-11-10 18:57 file2
$
```

此时，可以利用mv命令，重新命名文件，但不能采用问号“?”通配符，否则系统会误以为用户想把两个文件移至某个目录下。为此，可以借助于Ctrl-V组合键，在相应的字符位置上输入控制字符，即可解决文件的改名问题：

```
$ mv fi^V^Ble3 file3
$ mv fi^V^Tle3 file4
$ ls -l
总计 16
-rw-r--r-- 1 qqxing qqxing 1663 2009-11-10 18:56 file1
-rw-r--r-- 1 qqxing qqxing 2556 2009-11-10 18:57 file2
-rw-r--r-- 1 qqxing qqxing 1268 2009-11-10 19:00 file3
-rw-r--r-- 1 qqxing qqxing 1998 2009-11-10 18:57 file4
$
```



Ctrl-V是一个特殊的控制字符，在需要输入各种控制字符或其他特殊字符时，可以先按下Ctrl-V组合键，然后接着输入想要输入的任何控制字符，即可达到目的。从某种意义上说，Ctrl-V相当于一个转义符号，可用于引入任何控制字符。

5.2.4 递归地列出文件

利用ls命令的“-R”选项，可以递归地逐层显示当前（或指定）目录及其子目录下的所有文件。示例如下：

```
$ cd /etc/power
$ ls -lR
.:
总计 8
```

```
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 event.d
drwxr-xr-x 2 root root 4096 2009-10-29 05:56 scripts.d

./event.d:
总计 4
-rwxr-xr-x 1 root root 811 2009-10-07 07:02 laptop-mode

./scripts.d:
总计 4
-rwxr-xr-x 1 root root 874 2009-10-07 07:02 laptop-mode
$
```



“ls -l *”与“ls -lR”两个命令的意义和输出结果也不相同。前者仅仅涉及当前目录及其直接子目录两个目录层次中的所有文件。而后者将会遍历当前目录及其所有子目录（包括子目录下的子目录）中的所有文件。

5.3 显示文件内容

5.3.1 使用cat命令显示文件

传统的文件显示命令是cat，其语法格式简写如下：

```
cat [-sv] [file]
```

其中，“-s”选项表示把连续的多个空行压缩成一个空行输出。“-v”选项表示以“^X”或“M-”的形式显示文件中的不可打印字符（如果存在），但换行符和制表符等除外。

cat命令用于连续地显示文件的内容。不管终端屏幕的窗口有多大，cat命令总是从头到尾显示整个文件的所有内容。cat命令的缺点是：如果文件太大、太长，最终能够见到的只是文件的最后一部分内容，之前的内容将会快速闪过。例如，为了显示passwd文件，可以输入下列命令：

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
.....
$
```

cat命令的另一个常见用法是合并两个或多个文件，最终组成一个大的文件。实际上，实现文件合并的方式还有很多种。例如，可以使用下列命令把分章编写的文件合并为一个完整的文件：

```
$ cat chap1 chap2 chap3 chap4 chap5 > user_guide
$ ls
chap1 chap2 chap3 chap4 chap5 user_guide
$
```

如果要把两个文件的内容合并到其中的一个文件中，可以使用cat命令和I/O重定向(>)功能，把第二个文件中的内容附加到第一个文件中：


```
cat file2 >> file1
```

其中，第二个文件file2的输出被重定向，附加到第一个文件file1的后面，最终结果是把第二个文件的内容合并到第一个文件中。



如果采用下列命令形式合并文件，将会清除file1的数据内容，仅仅把file2的数据内容复制到file1中：

```
cat file1 file2 > file1
```

5.3.2 使用more命令分页显示文件

若想从头到尾一页一页地仔细阅读文件，可以使用more命令，逐页逐屏地显示整个文件的内容。more命令的语法格式简写如下：

```
more [options] [file]
```

例如，如果想要逐页显示/etc/profile文件，可以输入下列命令（其输出结果如图5-2所示）：

```
$ more /etc/profile
```

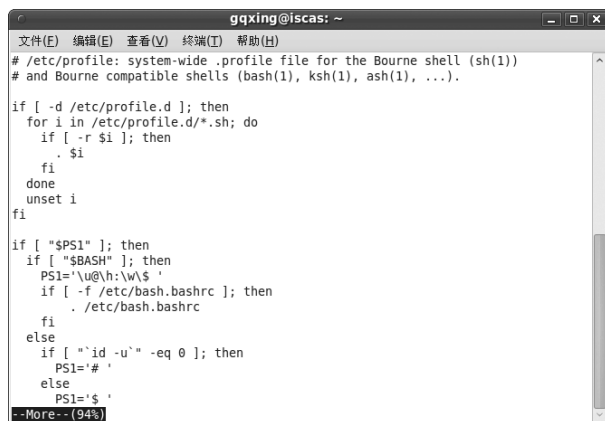


图5-2 more命令的输出结果

more命令在显示一页文件内容之后，屏幕窗口的左下角将会出现“--More--(n%)”信息，其中的“n%”表示已显示的数据内容占整个文件的百分比（如果文件很短，上述信息将不会出现）。在more命令的控制下，可以使用下列字符命令或控制键，前后滚动终端窗口，逐页逐屏地查看文件：

- `SPACE`：显示下一页，或显示下“i”行（如果指定行数后，接着按下空格键）。
- `Enter`：显示下一行，或显示下“i”行（如果指定行数后，接着按下Enter键）。
- `b (^B)`：回显前一页，或回显前第“i”页（如果指定页数后，接着输入字符“b”或按下Ctrl-B组合键）。
- `d (^D)`：显示下半页（初始值为11行），或显示下“i”行（如果指定行数后，接着输入字符“d”或按下Ctrl-D组合键）。
- `f`：显示下一页，或第“i+1页”（如果指定页数后，接着输入字符“f”）。
- `s`：跳过指定的行数之后，接着显示下一页。

- `␣`: 重新显示终端窗口中原有的数据内容。
- `v`: 调用默认的vim编辑器, 编辑当前文件, 并把光标定位在当前行。退出vim后再返回more命令的原位置。
- `i/pattern`: 从当前位置开始, 检索下一个或下面第“i”个匹配给定模式的字符串。
- `!command`: 调用Shell执行指定的命令。
- `:n`: 显示下一个文件(按照命令行列举的文件名顺序)。
- `:p`: 显示前一个文件(按照命令行列举的文件名顺序)。
- `:f`: 显示当前文件的名字与行号。
- `=`: 显示当前的行号(数)。
- `.`: 重复执行前一个命令。
- `h(?)`: 给出more命令的简要说明。
- `q(Q)`: 退出more命令。



注意

这里所谓的下一页或下一行意指文件结尾方向, 前一页或前一行意指文件开始方向。“i”的默认值为1。

5.3.3 使用less命令分页显示文件

Linux系统还提供一个与more类似的less命令, 其功能比more命令更强, 也能够分页显示文件。less命令的语法格式简写如下:

```
less [options] [file]
```

例如, 为了逐页地显示/etc/inputrc文件, 可以输入下列less命令, 其输出结果如图5-3所示

```
$ less /etc/inputrc
```

```
gqxing@iscas: ~
文件(E) 编辑(E) 查看(V) 终端(T) 帮助(H)
# /etc/inputrc - global inputrc for libreadline
# See readline(3readline) and 'info rluserman' for more information.

# Be 8 bit clean.
set input-meta on
set output-meta on

# To allow the use of 8bit-characters like the german umlauts, uncomment
# the line below. However this makes the meta key not work as a meta key,
# which is annoying to those which don't need to type in 8-bit characters.

# set convert-meta off

# try to enable the application keypad when it is called. Some systems
# need this to enable the arrow keys.
# set enable-keypad on

# see /usr/share/doc/bash/inputrc.arrows for other codes of arrow keys

# do not bell on tab-completion
# set bell-style none
# set bell-style visible

/etc/inputrc
```

图5-3 less命令的输出结果

首次进入less命令, 并在显示第一页文件内容之后, less命令将会在终端窗口的左下角显示当前文件的名字。当使用下列命令, 逐页逐屏地滚动翻阅文件后, less命令将会在终端窗口的左下角显示一个冒号“:”提示符。在less命令的控制下, 用户可以使用下列字符命令或控制键, 前后滚动终端窗口, 逐页查阅文件。

- `iSPACE`: 显示下一页, 或显示下 “i” 行 (如果指定行数后, 接着按下空格键)。
- `if (i'F)`: 同上。
- `^V`: 显示下一页。
- `iEnter`: 显示下一行, 或显示下 “i” 行 (如果指定行数 “i” 后, 接着按下Enter键)。
- `i'N`: 同上。
- `ie (i'E)`: 同上。
- `ij (i'J)`: 同上。
- `id (i'D)`: 显示下半页, 或显示下 “i” 行 (如果指定行数 “i” 后, 接着输入字符 “d” 或按下Ctrl-D组合键)。
- `ib (i'B)`: 回显前一页, 或回显前 “i” 行 (如果指定行数 “i” 后, 接着输入字符 “b” 或按下Ctrl-B组合键)。
- `iu (i'U)`: 回显前半页, 或回显前 “i” 行 (如果指定行数 “i” 后, 接着输入字符 “u” 或按下Ctrl-U组合键)。
- `ir (i'R)`: 重新显示屏幕中原有的数据内容。
- `^L`: 同上。
- `:n`: 显示下一个文件 (按照命令行列举的文件名顺序)。
- `:p`: 显示前一个文件 (按照命令行列举的文件名顺序)。
- `:f`: 显示当前文件的名字与行号等信息。
- `/pattern`: 从当前位置开始, 往下检索匹配给定模式的字符串。
- `?pattern`: 从当前位置开始, 往前检索匹配给定模式的字符串。
- `!command`: 调用Shell执行指定的命令。
- `h (H)`: 给出less命令的简要说明。
- `q (Q)`: 退出less命令。

同样, 这里所谓的下一页或下一行意指文件结尾方向, 前一页或前一行意指文件开始方向。“i” 的默认值为1。

5.3.4 显示文件前几行内容

有时, 只需查看文件的前几行内容, 即可对文件有一个清楚的了解, 此时可以使用head命令, 其语法格式简写如下:

```
head [-number | -n number] [file]
```

其中, number是一个数字, 表示需要输出的行数。在默认情况下, head命令将显示给定文件的前10行内容 (包括空行), 例如:

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
```

```
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
$
```

5.3.5 显示文件最后几行内容

对于比较大的日志文件，有时只想查看文件后面的最新信息。此时需要用到tail命令，其语法格式简写如下：

```
tail [ ±number [-lbcf]] [file]
```

其中，加号“+”表示从文件的起始位置开始计算。减号“-”表示从文件的结束位置开始计算。number是一个数字，表示需要输出的行数。在默认情况下，tail命令将会显示给定文件的最后10行（包括空行）的数据内容，例如：

```
$ tail /var/log/syslog
Nov 20 16:18:00 iscas kernel: [ 48.608071] i915 0000:00:02.0: LVDS-1: no EDID data
Nov 20 16:18:00 iscas kernel: [ 48.693153] [drm] DAC-6: set mode 640x480 0
Nov 20 16:18:00 iscas kernel: [ 49.186838] [drm] DAC-6: set mode 640x480 0
Nov 20 16:18:01 iscas kernel: [ 49.668063] i2c-adapter i2c-1: unable to read EDID block.
Nov 20 16:18:01 iscas kernel: [ 49.668071] i915 0000:00:02.0: LVDS-1: no EDID data
Nov 20 16:18:01 iscas kernel: [ 49.779450] [drm] DAC-6: set mode 640x480 0
Nov 20 16:18:02 iscas kernel: [ 50.272245] [drm] DAC-6: set mode 640x480 0
Nov 20 16:18:02 iscas kernel: [ 50.756064] i2c-adapter i2c-1: unable to read EDID block.
Nov 20 16:18:02 iscas kernel: [ 50.756071] i915 0000:00:02.0: LVDS-1: no EDID data
Nov 20 16:18:10 iscas wpa_supplicant[636]: CTRL-EVENT-SCAN-RESULTS
$
```

上述tail命令只能查看静态文件的最后几行内容，对于不断增长的日志文件，有时需要持续地监控文件不断出现的最新信息。此时可以使用带有“-f”选项的tail命令。例如，为了监控某个日志文件的最新变化，可以使用下列命令：

```
tail -f somelogfile
```

5.4 复制文件

在开发程序时，经常需要复制以前的程序，并以此为基础进行修改，这时就要用到文件的复制命令cp。cp命令的语法格式简写如下：

```
cp [-ir] source_file target_file
```

其中，“-i”选项表示交互复制方式。当指定的目的文件存在时，cp命令将会提示用户存在同名的文件。仅当用户输入y确认之后，cp命令才会继续执行复制；其他任何回答将会终止复制的执行。“-r”选项表示递归复制方式。当指定的源文件为目录时，cp命令将会递归地复制目录及其中的任何文件，包括子目录及其中的文件。

假定我们想利用先前创建的新文件复制一个新文件，可以使用下列命令：

```
$ cp newfile newcopy
$
```

此时，当前目录下将会增加一个新复制的文件，利用ls命令进行检验，其结果如下：

```
$ ls -l
总计 8
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:35 newcopy
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:33 newfile
$
```

使用cp命令不仅可以复制当前目录下的文件，也可以复制其他目录中的文件。例如，下列命令将会把/etc目录下的profile文件复制到当前目录，文件名保持不变：

```
$ cp /etc/profile .
$
```

值得注意的是，在使用cp命令而不加任何选项时，如果访问权限许可，有可能误删已有的同名文件，将同名的文件覆盖，造成不应有的损失。因此，比较保险的做法是在使用cp命令时，有意识地增加“-i”选项。使用“-i”选项时，如果目的文件不存在，cp命令将会正常地执行。如果目的文件存在，系统将会输出提示信息，请求用户予以确认。示例如下：

```
$ cp -i somefile somefile2
cp: 是否覆盖 “somefile2” ?
$
```

此外，利用“-r”选项，cp命令还能够复制整个目录（包括目录中的所有子目录及文件），详情参见5.11节。

5.5 移动文件

使用mv命令，可以把文件从一个目录移到另外一个目录中，或重新命名一个文件。mv命令的语法格式简写如下：

```
mv [-f] source_file target_file
```

其中“-f”选项为强制移动或改名，“-i”选项意味着，一旦目标目录或文件存在，在取得用户的认可后才能采取下一步的处理动作。下面是一个利用mv命令把newfile文件移至/tmp目录，而文件名保持不变的例子：

```
$ ls -l
总计 8
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:35 newcopy
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:33 newfile
$ mv newfile /tmp
$ ls -l
总计 4
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:35 newcopy
$ ls -l /tmp/newfile
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:33 /tmp/newfile
$
```

下面的例子则利用mv命令把newcopy文件改名为oldcopy：

```
$ mv newcopy oldcopy
$
```

再次使用ls命令，可以看到原来的文件已经不存在了，newcopy已改名为oldcopy：

```
$ ls -l
总计 4
-rw-r--r-- 1 gqxing gqxing 22 2009-11-10 18:35 oldcopy
$
```

另外，mv命令也存在一个潜在的问题，即当新命名的文件已经存在时，如果访问权限许可，同名的文件将被覆盖。为了防止此类情况的出现，可在mv命令中使用“-i”选项，一旦同名文件存在，mv命令将会提示用户，且仅在用户认可的情况下才会覆盖原有的文件，从而避免出现误删文件的问题，例如：

```
$ mv -i file1 file2
mv: 是否覆盖 “file2” ?
$
```

为了重新命名一个文件，还有一个比较有用的命令basename。basename命令的本意是剔除文件名中的目录部分，也即文件路径名中最后一个斜线字符“/”之前的所有目录前缀，使之仅包含文件名本身，或删除文件名的扩展名后缀。其语法格式简写如下：

```
basename string [suffix]
```

假定有一个C程序/home/gqxing/src/atmcom.c，使用下列命令即可取出删除了目录路径名和“.c”后缀的文件名atmcom：

```
$ basename /home/gqxing/src/atmcom.c .c
atmcom
$
```

为了编译任何C程序，把编译后的a.out文件改名为相应的程序文件，可以创建下列Shell脚本（注意，命令前后加反向单引号“`”表示用命令的输出结果替换当前位置的内容，参见第7章“Shell基础知识”的“命令替换”一节）：

```
$ cat comp
gcc $1
mv a.out `basename $1 .c`
$
```

其中的\$1是位置参数，可以是提供给comp脚本的任何一个C程序文件（参见第7章“Shell基础知识”）。执行下列命令后即可编译atmcom.c源程序，把编译后的a.out程序文件改名为atmcom：

```
$ cd /home/gqxing/src
$ comp atmcom.c
$
```

再如，为了把扩展名为“.old”的文件改名为扩展名为“.c”的C源程序文件，可以使用“*.old”作为参数，运行下列Shell脚本：

```

$ cat rename
for i in $*
do
    mv $i `basename $i .old`.c
done
$ ls -l *.old
-rw-r--r-- 1 qqxing qqxing 30235 2009-11-17 17:56 atmcom.old
-rw-r--r-- 1 qqxing qqxing 82468 2009-11-17 18:04 handler.old
-rw-r--r-- 1 qqxing qqxing 58257 2009-11-17 18:02 listener.old
$ rename *.old
$ ls -l *.c
-rw-r--r-- 1 qqxing qqxing 30235 2009-11-17 17:56 atmcom.c
-rw-r--r-- 1 qqxing qqxing 82468 2009-11-17 18:04 handler.c
-rw-r--r-- 1 qqxing qqxing 58257 2009-11-17 18:02 listener.c
$

```

5.6 删除文件

如果需要删除文件，可使用rm命令。rm命令的语法格式简写如下：

```
rm [-rfi] [file]
```

其中，“r”选项用于递归地删除目录中的文件及目录本身，参见5.12节。“i”选项表示以交互方式执行文件删除操作。在删除任何文件之前，rm命令将会请求用户予以确认。“f”选项表示强制删除文件。此外，即使文件不存在，rm命令也不会输出任何错误信息。

在Linux系统中，文件一旦删除，很难再恢复。因此，在执行文件删除操作时一定要小心谨慎。在使用rm命令时，建议增加“-i”选项，使得在真正删除文件之前还有一次选择的机会。下面以newcopy文件为例，说明怎样使用rm命令删除文件。

```

$ ls -l
总计 8
-rw-r--r-- 1 qqxing qqxing 22 2009-11-10 18:35 newcopy
-rw-r--r-- 1 qqxing qqxing 22 2009-11-10 18:33 newfile
$ rm newcopy
$ ls -l
总计 4
-rw-r--r-- 1 qqxing qqxing 22 2009-11-10 18:33 newfile
$

```

如果使用“-i”选项删除文件，其情况如下：

```

$ rm -i newcopy
rm: 是否删除普通文件 “newcopy” ?
$

```

如果想要一次删除多个文件，可以把文件名一一列在rm命令之后，也可以使用通配符，例如

```

$ rm -i new*
rm: 是否删除普通文件 “newfile” ? n
rm: 是否删除普通文件 “newcopy” ? y
$

```



使用星号“*”通配符时要小心谨慎，尤其是下列用法更要慎之又慎：

```
$ rm *  
$
```

如果对删除操作确有把握，需要强制删除某些文件时（这在部分系统维护脚本中尤为常见），可以使用“-f”选项，强制删除指定的文件，例如：

```
$ rm -f *.tmp  
$
```

5.7 显示当前工作目录

在Bash环境中，命令提示符中通常会包含当前工作目录最低一级的子目录，当需要确定自己当前所处的准确目录位置时，可以使用pwd命令。pwd命令主要用于显示当前的工作目录，以便用户随时了解自己在文件系统目录层次中所处的位置。例如：

```
$ pwd  
/usr/include  
$
```

实际上，通过设置PS1内部变量，也可以在命令提示符中输出完整的当前工作目录（参见第9章“用户管理”）。

5.8 改换目录

在Linux系统中，每个用户都有一个属于自己的主目录。在注册之后，系统将会自动地把用户引导至自己的主目录。当需要转入某个工作目录时，可使用cd命令。cd（改换目录）命令使用户能够进入文件系统的任何目录层次（除非目录的访问权限不允许）。例如：

```
$ cd /usr/lib  
$ pwd  
/usr/lib  
$
```

cd命令的默认参数为\$HOME，即当前用户的主目录。因此，当输入一个未加任何命令选项与参数的cd命令时，相当于执行“cd \$HOME”命令，系统将会把当前的工作目录改换到用户的主目录。例如，如果用户的主目录是/home/gqxing，输入下列命令即可返回自己的主目录：

```
$ cd  
$ pwd  
/home/gqxing  
$
```

在Bash、Korn Shell以及TCShell等Shell中，波浪号“~”可用做用户主目录的缩写符号。例如，使用下列命令，可把工作目录改换到当前用户主目录下的music子目录：

```
$ cd ~/music  
$
```


也可以使用下列缩写形式, 指定任何一个用户的主目录。其中, `username` 可以是任何注册的用户名:

```
cd ~username
```

在不支持波浪号“~”缩写形式的Shell (如Bourne Shell) 中, 引用用户主目录的替代方法是引用`$HOME`变量, 这也是每个Shell都支持的通用方法。例如, 可以使用下列命令, 把工作目录改换到当前用户主目录下的`script`子目录:

```
$ cd $HOME/script
$
```

如前所述, 句点“.”表示当前目录, 双句点“..”表示父目录。为了从子目录返回父目录, 可使用“`cd ..`”命令。

```
$ pwd
/home/gqxing/script
$ cd ..
$ pwd
/home/gqxing
$
```

假定当前的工作目录是`/home/user1`, 现准备转到`/home/user2`目录下处理其中的文件, 可以使用下列命令:

```
$ pwd
/home/user1
$ cd ../user2
$ pwd
/home/user2
$
```

在使用`cd`命令时, 可以指定绝对目录名, 也可以使用相对目录名。所谓绝对目录名, 是指从文件系统的根目录“/”开始, 按层次结构逐级列出每一级子目录, 直至最终子目录的路径名。所谓相对目录, 是指相对于当前目录的路径名。

在上述例子中, “`cd ../user2`”引用的就是相对目录。如果改用下列“`cd /home/user2`”命令, 其中引用的就是绝对路径:

```
$ pwd
/home/user1
$ cd /home/user2
$ pwd
/home/user2
$
```

5.9 创建目录

许多用户习惯于创建不同的目录, 分类存储各种不同的文件。为了创建新的目录, 可以使用`mkdir`命令, 同时在命令后面指定新建目录的名字。例如:

```
$ mkdir src
$ cd src
$ mkdir java
$ mkdir c
$ cd java
$ pwd
/home/gqxing/src/java
$ cd ..
$ ls -l
总计 8
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:57 c
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:58 java
$
```

5.10 移动目录

同文件操作一样，也可以利用mv命令，把目录从一个位置移动到另一个位置。例如，利用下列mv命令，可以把src目录下的java子目录移至上一级目录，即移至主目录：

```
$ cd /home/gqxing/src/java
$ mv java ..
$ cd ..
$ ls -l
总计 16
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:58 incl
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:58 java
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:58 script
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-12 18:58 src
$
```

同样，也可以利用mv命令重新命名一个目录，为目录改换一个名字：

```
$ mv script shell
$ ls -l shell
总计 16
-rwxr-xr-x 1 gqxing gqxing 919 2009-11-12 18:58 ftpget
-rwxr-xr-x 1 gqxing gqxing 602 2009-11-12 18:58 prime
-rwxr-xr-x 1 gqxing gqxing 789 2009-11-12 18:58 tree
-rwxr-xr-x 1 gqxing gqxing 640 2009-11-12 18:58 whatday
$
```

5.11 复制目录

为了把某个目录下的所有文件全部移至另外一个目录，可以使用cp命令的“-r”选项实现。例如：

```
$ cp -r dir1 dir2
$
```

cp命令的“-r”选项意味着递归复制。执行上述命令后，源目录dir1下的所有文件、子目录及其子目录下的所有文件将全部复制到目的目录dir2中。在复制目录时，如果不加“-r”选项，

cp 命令将会输出一个错误信息。

另外，还可以利用cpio和后面即将介绍的find命令实现目录的复制。通过管道组合使用这两个命令，可以构成一个功能非常丰富的复合命令。cpio命令可用于备份任何目录或文件系统中的文件，把其中的文件归档后复制到任何存储介质中。它的另外一个重要功能就是能够把位于指定目录下的所有目录文件依其原有的层次结构原封不动地复制到另一个目录中，其语法格式简写如下：

```
cpio -p [-adlLmuvV] [-R id] directory [< fname_list]
```

其中，“-p”选项表示从标准输入中读取路径文件名，然后按照其他选项的要求，把指定目录中的目录和文件按原有的层次结构复制到目的目录中。表5-2给出了其他命令选项的说明。

表5-2 find命令的其他部分选项及其简单说明

选项	GNU选项	说明
-a	--reset-access-time	完成文件复制后恢复源文件的访问时间属性，使得cpio命令的文件访问不留痕迹
-d	--make-directories	根据源目录层次结构的要求创建目录
-m	--preserve-modification-time	使复制的目的文件保持源文件原有的修改时间不变（但对新创建的目录无效）
-u	--unconditional	无条件的复制。通常，时间较早的老文件不能替换与复制目的文件同名的新文件
-v	--verbose	显示cpio命令处理的文件列表及其属性信息

例如，为了把dir1目录中的所有子目录及文件原封不动地复制到一个新目录dir2中，可使用下列组合命令：

```
$ cd dir1
$ find . -print | cpio -padmuv dir2
.....
$
```

5.12 删除目录

若想删除一个空的目录，可以使用rm dir命令，其语法格式简写如下：

```
rmdir [-p] directory
```

如果目录中包含文件，使用rm dir命令删除目录时将会出错。此时可以使用带有“-r”选项的rm命令。利用“-r”选项删除目录是一种递归操作，可以把指定目录及其任何子目录中的所有文件全部删除，例如：

```
$ rm -r dir2
$
```



使用“rm -r”命令删除目录及其文件时，一经提交命令，通常是无法挽回的。因此，使用这个命令时必须谨慎行事。

5.13 比较文件之间的差别

当面对两个类似的文件，想找出其中的细微差别时，可以使用diff命令进行比较，从中找出两个文件的不同之处。diff命令的语法格式简写如下：

```
diff [options] file1 file2
```

diff命令分别读取两个输入文件，逐行分析其中的异同点，从而找出两者之间的差别。当找出不同的行时，diff命令将会尝试确定出现差别的行是否是由于插入、删除或修改文本行等原因造成的，如果确实如此，还要检查有多少行受到影响。最后，diff命令将会告诉用户，每个文件的哪一行，从哪个字符开始，有几个字符不同，同时给出两个文件中存在差别的文本行。

如果两者之间的差别是由于插入新的文本行造成的，diff命令将会采用下列形式显示新增的行：

```
lline#1[,lline#2] a rline#1[,rline#2]
```

其中，lline#1和选用的lline#2是第一个文件中的行号，rline#1和选用的rline#2是第二个文件中的行号。

如果两者之间的差别是由于删除文本行造成的，diff程序将会采用下列形式显示哪一个文件删除了文本行：

```
lline#1[,lline#2] d rline#1[,rline#2]
```

如果两者之间的差别是由于修改文本行造成的，diff程序将会采用下列形式显示发生变更的文本行：

```
lline#1[,lline#2] c rline#1[,rline#2]
```

在上述的任何情况下，两个文件中的相关文本行将会随行号一并给出。第一个文件中的文本行前面冠以小于号“<”，第二个文件中的文本行前面冠以大于号“>”。

假定有两个文件test1和test2，其中的数据内容分别如下：

```
$ cat test1
You are in a maze of
twisty little passages
which are all alike.
$ cat test2
You are in a maze of
twisty little passages
which are all different.
$
```

利用diff命令进行比较，其输出结果如下：

```
$ diff test1 test2
3c3
< which are all alike.
---
> which are all different.
$
```

其中的“3c3”表示两个文件的第3行数据内容不同，不同的原因是由于部分数据内容的变动造成的。

如果两个文件完全相同，diff命令将不会显示任何信息。有关diff命令的详细说明和其他功能，可以查阅Linux系统的随机文档。

5.14 从系统中检索文件

当用户想要找出具有某一特征的文件，或了解系统中是否存在某个文件，又不知文件究竟在哪个目录时，可以使用find命令。

find命令将按用户指定的检索条件，从指定的目录开始，找出满足匹配准则的所有文件。指定的检索条件可以是文件名（包括通配符）、文件大小，以及文件修改日期等。find命令的一般语法格式简写如下：

```
find directory [options]
```

其中，directory是检索的起始目录，options是一种表达式选项，用于指定各种匹配准则或检索条件。每个选项均描述一种文件匹配或检索准则。一个文件必须满足所有的匹配或检索原则才能选中。使用的选项越多，选中的文件越少。表5-3列出了find命令的部分常用选项及其简单说明。

表5-3 find命令的部分常用选项

选项	简单说明
-name <i>filename</i>	检索匹配指定文件名的所有文件。其中，指定的文件名不必包括目录路径。也就是说，只要文件名本身匹配即可。例如，如果指定的文件名为hosts，则/etc/hosts与/etc/avahi/hosts等均为匹配检索准则的文件。如果指定的文件名中包括通配符“*”、“?”和“[...]”，文件名前后应加单引号或双引号
-user <i>username</i>	检索其文件属主匹配指定用户的所有文件。其中，username可以是任何一个合法的注册用户名，也可以是用户ID号
-group <i>groupname</i>	检索其用户组属性匹配指定用户组的所有文件。其中，groupname可以是任何一个合法的用户组名，也可以是用户组ID号
-nouser	检索其文件属主并未在/etc/passwd文件中定义的所有文件，也即检索其文件属主非本地系统用户的文件
-nogroup	检索其用户组名并未在/etc/group文件中定义的所有文件，也即检索其用户组非本地系统用户组的文件
-atime [<i>±</i>] <i>n</i>	选择在 <i>n</i> 日之前、之内或恰好 <i>n</i> 天访问过的文件
-ctime [<i>±</i>] <i>n</i>	选择在 <i>n</i> 日之前、之内或恰好 <i>n</i> 天状态信息发生变动的文件
-mtime [<i>±</i>] <i>n</i>	选择在 <i>n</i> 日之前、之内或恰好 <i>n</i> 天修改过文件内容的文件
-newer <i>filename</i>	选择其修改日期比给定文件更近的文件
-depth	表示逐层深入各级子目录，采用先文件后目录的方式，自底向上依次检索所有的文件和目录

(续表)

选项	简单说明
<code>-size [±]n[cwbkMG]</code>	按照指定的文件大小数值n检索符合条件的文件。其中，n通常表示以512个字节数据块为单位的文件大小。如果n后面附加一个字符c、w、b、k、M或G，则分别表示指定的文件大小以字节、双字节的字、512字节的数据块（默认值）、1KB、1MB或1GB为计数单位。其中： <ul style="list-style-type: none"> · +n: 表示大于指定的数量 · n: 表示恰好等于指定的数量 · -n: 表示小于指定的数量
<code>-inum n</code>	检索匹配指定信息节点号的文件
<code>-type filetype</code>	检索指定类型的文件。其中的文件类型可以是： <ul style="list-style-type: none"> · f: 表示普通文件 · d: 表示目录 · b: 表示块特殊文件 · c: 表示字符特殊文件 · p: 表示管道（FIFO）文件 · l: 表示符号链接文件 · s: 表示套接字文件
<code>-perm [±]mode</code>	检索匹配指定访问权限（以八进制数值或符号形式表示）的文件。如果mode字段之前存在一个减号“-”，表示文件的访问权限必须包括mode定义的所有访问权限。如果mode字段之前为加号“+”，表示文件的访问权限至少必须包括mode定义的一种访问权限。如果mode字段之前没有加减号，表示文件的访问权限必须完全匹配mode定义的所有访问权限
<code>-perm /mode</code>	其功能类似于“-perm +mode”，用于检索匹配指定访问权限（以八进制数值或符号形式表示）的文件
<code>-links [±]n</code>	检索其链接计数大于、等于或小于指定数量n的文件
<code>-exec cmd {} \; [+]</code>	把find命令的检索结果作为参数提交给指定的命令，由给定的命令做进一步的加工处理。后面的花括号表示给定命令的参数将由find命令的输出结果予以替换。命令的后面必须以转义的分号“\;”或转义的加号“\+”结束 当命令以加号结束时，意味着把find命令的输出结果汇总为一个参数集合，然后一次性地提交给定的命令。因此，使用加号“+”而非分号“;”能够改善命令的运行性能
<code>-ok cmd ;</code>	其功能类似于“-exec”选项，唯一的差别是在执行给定的命令之前输出请求信息，当且仅当用户输入“y”（或“Y”）确认之后才继续执行
<code>-empty</code>	文件为空，且为普通文件或目录
<code>-ls</code>	以“ls -dils”命令的输出格式输出匹配的文件，文件的大小以1K字节的数据块为单位，除非环境变量PCSIPLY_CCRRECT已经设置（表示按512字节的数据块计算文件的大小）
<code>-print</code>	打印检索结果，即输出符合检索条件的文件名。这个选项是find命令默认的处理动作，可以省略

5.14.1 简单检索

1. 按文件名模式检索文件

假定想要检索当前工作目录及其子目录下所有以“.c”为后缀的C程序文件，可输入下列命令：

```
$ find . -name '*.c' -print
./src/atmmon.c
./src/listener.c
.....
$
```

上述find命令中的句点“.”表示当前目录。这意味着从当前目录开始，遍历当前目录及其子目录，检索匹配“-name”选项（“*.c”）指定的所有C源程序文件。



当文件名匹配模式中包含通配符时，一定要用单引号或双引号括起来，以便Shell能够正确地解释。

2. 按照文件的日期属性检索文件

利用“-atime”、“-ctime”或“-mtime”选项，可以根据文件的访问日期、创建日期或修改日期检索文件。例如，利用下列命令可以查询当前目录中最近10天之内修改过的文件：

```
$ find . -mtime -10
```

假定我们还想在/home/gqxing目录及其子目录中找出在创建了某个标志文件（myfile）之后创建、访问或修改过的所有文件，可以使用下列命令：

```
$ find /home/gqxing -newer myfile
```

5.14.2 使用逻辑运算符

find命令允许用户使用逻辑非“!”、逻辑与“-a”和逻辑或“-o”等逻辑运算符组合各种选项，定义更为严格的检索准则。在使用逻辑表达式时，组合选项前后要加一对转义的圆括号。

如果想要检索不符合某个特定属性选项定义的文件，可以使用逻辑非运算符“!”。例如，假定我们打算找出/etc目录中所有不属于用户root的文件，可以使用下列命令：

```
$ find /etc ! -user root
```

如果想要检索同时具有两种不同属性的文件，可使用逻辑与运算符“-a”定义组合选项。例如，假定我们准备找出根目录中属于gqxing用户的所有子目录，可以使用下列命令：

```
$ find / -type d -a -user gqxing
```

如果想要检索具有其中的一种或两种属性的文件，可以使用逻辑或运算符“-o”定义组合选项。例如，假定我们期望检索系统中一个月来从未访问过的，文件扩展名为“.o”或文件名为a.out的所有文件，可以使用下列命令：

```
$ find / \( -name '*.o' -o -name a.out \) -atime +30
```

5.14.3 调用其他命令处理检索结果

利用find命令的“-exec”选项，还可以采用下列两种命令形式，以批处理的方式，把检索出来的文件作为参数，交由其他命令做进一步的加工处理：

```
-exec command {} \;
-exec command {} \+
```

其中，`command`可为任何文件处理命令，花括号表示其参数取自`find`命令的输出，即由`find`命令检索出来的文件名予以替换。注意，“`-exec`”选项的后面必须附加转义的分号“`\;`”或转义的加号“`\+`”，作为命令的终止符。

例如，如果想要删除当前目录及其子目录中扩展名为“`.tmp`”的所有文件时，可以使用下列命令：

```
$ find . -name '*.tmp' -exec rm {} \;
$
```

假定因为某种原因，使得一个目录（及其各级子目录）中所有文件的文件属主或访问权限发生了变化，导致应用无法正常运行时，可以利用`find`命令列出所有的文件，由`chown`或`chmod`等命令予以修正，例如：

```
$ find . -type f -exec chown gqxing '{}' \;
$
```

5.14.4 利用管道实现其他处理功能

通常，Linux命令能够接受的参数具有一定的限量，虽然这个限量一般情况下并不妨碍我们执行任何Linux命令，但当使用上述`find`命令形式进行批处理时，`find`命令的输出有时是很可观的，“`-exec`”选项后面的命令可能无法接受如此多的参数。为了解决这个问题，一种常见的替代做法是利用管道，将`find`命令的输出提交给`xargs`命令协助执行。

例如，为了把某个目录下所有普通文件的访问权限改为644，所有子目录的访问权限改为755，可以使用下列两个命令实现：

```
find pathname -type f -print | xargs chmod 644
find pathname -type d -print | xargs chmod 755
```

把`find`命令的输出通过管道提交给`xargs`，由`xargs`命令协助执行，除了上述原因之外，还有性能方面的考虑。当把大量的文件和目录交由“`-exec`”选项后面的单个命令处理时，对每个文件或目录的处理都需要创建一个进程，因而需要创建太多的进程。而Linux系统允许每个用户创建进程的数量是有一定限制的。`xargs`的做法是把文件和目录名收集在一起，组成多个参数提交给单个Linux命令执行。这种处理方式只需创建较少的进程，因而能够提高命令的运行速度，改善系统的性能。

但是，当`find`命令的输出并没有多到无法接受时，如果在`find`命令的“`-exec`”选项后面直接采用转义的加号“`\+`”，在提高运行速度，改善系统性能方面，也能达到同样的目的，两者的效果基本上是一样的。

5.15 检索文件内容

本节主要讨论功能强大的文本检索工具`grep`。

5.15.1 利用`grep`检索文件内容

为了检索文件中的特定字符串，可以使用`grep`命令。`grep`命令的基本语法格式简写如下：


```
grep [-inv] string file
```

其中，“-i”选项表示在进行比较时忽略字母的大小写。“-n”选项表示在输出检索结果之前给出文本行在文件中的行号（行号从1开始计算）。“-v”表示检索不包含给定字符串或模式的所有文本行。*string*是一个检索模式。检索模式可以是一个准备检索的字符串、一个单词或短语。注意，检索模式可以包含空格、标点符号甚至控制字符，但需要在前后增加引号。*file*是准备检索的文件。

例如，为了从电话簿文件Phonebooks中检索John Smith的电话分机，可以使用部分或完整的名字进行模式匹配。示例如下：

```
$ grep Smith Phonebooks
John Smith      2810
$
```

如果检索模式是一个较长的字符串，由多个字组成，中间也可能包含空格字符，可以在字符串前后加单引号或双引号，例如：

```
$ grep "Louisa May" Phonebooks
Louisa May Alcott 2826
$
```

检索模式越简短，输出的冗余数据就越多。反之，检索模式的限制越多，符合检索条件的输出结果就越少，例如：

```
$ grep Al Phonebooks
Louisa May Alcott      2826
David Allan            2866
Edgar Allan Poe        2812
$ grep Allan Phonebooks
David Allan            2866
Edgar Allan Poe        2812
$
```

默认情况下，grep命令是严格区分大小写字母的，也就是说，在输入检索模式的字符串时必须注意字母的大小写，例如：

```
$ grep allan Phonebooks
$ grep Allan Phonebooks
David Allan            2866
Edgar Allan Poe        2812
$
```

上述的第一个grep命令之所以检索失败（没有给出检索结果），就是因为字母“a”的大小写拼写错误。

5.15.2 过滤其他命令的输出数据

grep命令的主要用途是从输入文件中获取感兴趣的数据，过滤掉不需要的数据内容。因此，通常把grep称做滤通程序或过滤程序。grep命令的常见用法是过滤其他命令的输出结果，从命令输出中抽取含有某种特征的数据。因此，必须采用管道机制，把命令的输出数据提交grep命令。

假定当前目录中存在一系列不同时间开发的C程序:

```
$ ls -l *.c
-rw-r--r-- 1 qqxing qqxing 833233 2009-11-10 16:22 buttons.c
-rw-r--r-- 1 qqxing qqxing 739245 2009-11-20 09:38 changes.c
-rw-r--r-- 1 qqxing qqxing 608368 2009-11-10 10:20 clock.c
-rw-r--r-- 1 qqxing qqxing 827114 2009-11-20 16:49 commands.c
.....
$
```

如果想要从中找出7月份开发的程序, 可以使用下列命令组合, 通过管道把ls命令的输出送交grep, 过滤后的输出结果如下:

```
$ ls -l *.c | grep 11-20
-rw-r--r-- 1 qqxing qqxing 739245 2009-01-20 09:38 changes.c
-rw-r--r-- 1 qqxing qqxing 827114 2009-01-20 16:49 commands.c
$
```

5.15.3 同时检索多个文件

grep命令可以同时检索多个文件。当找出匹配检索模式的字符串时, grep将会在输出信息前冠以文件的名字, 然后输出匹配检索模式的文本行。例如:

```
$ ls
beijing newyork shanghai washington
$ grep capital *
beijing:Beijing is the capital of China.
washington:Washington is the capital of the United States.
$
```

5.15.4 检索不包含特定模式的文本行

为了输出并不包含特定字符串的所有文本行, 可以使用grep命令的“-v”选项。下面的例子说明怎样在当前目录下的所有文件中找出不包含字符串capital的文本行。

```
$ grep -v capital *
newyork:New York is the biggest city in the United States.
shanghai:Shanghai is the biggest city in China.
$
```

5.15.5 使用正则表达式进行检索

在grep命令中, 也可以使用正则表达式作为检索模式, 检索匹配给定模式的字符串或文本行。正则表达式可以由普通字符、数字以及具有特殊意义的元字符组成。在grep的正则表达式中, 可用的元字符包括“/”、“\$”、“.”、“*”和“\”等, 如表5-4所示。注意, grep命令仅支持简单的正则表达式。欲实现复杂的模式匹配, 可以使用egrep等命令。

表5-4中的特殊字符对Linux系统也有特殊的意义。因此, 在grep命令中使用正则表达式时, 需要通过转义机制, 使系统在解释命令行期间忽略这些元字符的特殊含义。因此, 当用户在系统提示符下输入带有正则表达式的grep命令时, 应当使用引号括住正则表达式。

表5-4 grep检索模式可用的元字符

元字符	简单说明
<code>^</code>	匹配文本行的行首
<code>\$</code>	匹配文本行的行尾
<code>.</code>	匹配任何一个单字符
<code>[...]</code>	匹配字符集或字符范围中的任何一个字符
<code>[^...]</code>	匹配不属于字符集或字符范围中的任何一个字符
<code>*</code>	匹配零个或多个同一字符或正则表达式
<code>+</code>	匹配一个或多个同一字符或正则表达式
<code>\</code>	随后的元字符作为普通字符处理

此外，如果检索模式中包含元字符，且需要忽略其特殊意义时，可在元字符之前增加转义符号。

现在，我们将以下列五行数据（选自泰戈尔《Stray Birds》的两首诗）作为这一节的测试数据，说明如何在grep命令中使用正则表达式：

```
$ cat stray.birds
I cannot choose the best
The best choose me

Roots are the branches down in the earth
Branches are the roots in the air
$
```

由于元字符“`^`”表示行首，故下列命令将会找出输入文件中以字符“`T`”为行首字符的所有文本行：

```
$ grep '^T' stray.birds
The best choose me
$
```

为了仅仅列出当前目录中的子目录，可以使用下列命令：

```
$ ls -l | grep '^d'
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-10 22:45 bin
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-10 22:45 incl
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-10 22:45 script
drwxr-xr-x 2 gqxing gqxing 4096 2009-11-10 22:45 src
$
```

元字符“`$`”表示行尾，故下列命令将会找出输入文件中所有以字符“`t`”为行尾字符的文本行：

```
$ grep 't$' stray.birds
I cannot choose the best
$
```

下列命令可用于显示文件中只有一个字符“`b`”的文本行（输入文件中没有这样的文本行，故没有任何输出）：

```
$ grep '^b$' stray.birds
$
```

由于元字符 “.” 可以匹配任何一个字符，故下列命令能够匹配任何以 “an” 为头两个字符的三字符字符串，包括 “any”、“and”、“cannot” 和 “plan”（因为空格也计数）等：

```
$ grep 'an.' stray.birds
I cannot choose the best
Roots are the branches down in the earth
Branches are the roots in the air
$
```

为了列出当前目录中其他用户能够读写的文件，可以使用下列命令：

```
$ ls -l | grep '^-.....rw'
-rw-rw-rw- 1 gqxing gqxing 120 2009-11-10 22:32 stray.bird
$
```

为了找出文件中包含字符串 “the” 的文本行，且忽略大小写字母的区别，在输出数据前冠以行号，可以使用下列命令：

```
$ grep -i -n the stray.birds
1:I cannot choose the best
2:The best choose me
4:Roots are the branches down in the earth
5:Branches are the roots in the air
$
```

为了找出文件中的所有空行，并给出行号，可以使用下列任何一个命令：

```
$ grep -n ^$ stray.birds
3:
$ grep -n -v . stray.birds
3:
$
```

当检索模式（如字符、字符串或正则表达式）后面附加一个星号 “*” 字符时，grep 将把星号 “*” 解释为“重复匹配零个或多个模式”。

按照上述定义，检索结果也可以包含零个模式，这有可能使用户对星号的使用及输出结果感到困惑。下列命令或许可以稍做阐释，其中的检索模式 “ro*” 意味着找出字符串中含有一个字符 “r” 和零个或多个字符 “o”（如 “r”、“ro” 或 “roo” 等）的所有文本行：

```
$ grep 'ro*' list
Roots are the branches down in the earth
Branches are the roots in the air
$
```

在上述输出结果中，第一行匹配检索模式 “ro*” 的字符串是 “branches”，第二行匹配检索模式 “ro*” 的字符串是 “roots”。

同样，如果想要找出字符串中至少包含一个字符 “d” 的所有文本行，可以使用下列检索模式（其中第一个字符 “d” 仅表示字符文字 “d”，第二个字符 “d” 和 “*” 是一个正则表达式，表示零个或多个字符 “d”）：

```
$ grep 'dd*' stray.birds
Roots are the branches down in the earth
$
```

但如果想要找出字符串中至少包含两个字符“oo”的所有文本行，可以使用下列检索模式（其中的前两个字符“o”仅表示字符串“oo”，第三个字符“o”和“*”是一个正则表达式，表示零个或多个字符“o”）：

```
$ grep 'ooo*' stray.bird
I cannot choose the best
The best choose me
Roots are the branches down in the earth
Branches are the roots in the air
$
```

为了检索输入文件中匹配零个或多个任意字符的所有文本行，可以使用下列检索模式：

```
$ grep '.*' stray.bird
I cannot choose the best
The best choose me

Roots are the branches down in the earth
Branches are the roots in the air
$
```

由于句点“.”能够匹配任意一个字符，星号“*”能够重复匹配零个或多个模式，故“.*”能够匹配任何文本行，包括空行。

5.15.6 检索元字符本身

为了使用grep命令检索“&”、“!”、“.”、“*”、“?”和“\”等元字符本身，可以在元字符前面加转义符号“\”。转义符号能够使grep命令忽略元字符的特殊含义。例如，下列检索模式可以返回以句点“.”为起始字符的文本行。这在检索经过nroff或troff加工处理的文档时是非常有用的：

```
$ grep ^\.. somefile
```

5.15.7 在命令行中使用引号

正如先前所述，如果想把具有多个单词的短语作为一个字解释，可以使用引号把短语括起来。例如，为了从输入文件中检索短语“in the air”，可以使用下列grep命令：

```
$ grep "in the air" stray.bird
Branches are the roots in the air
$
```

此外，还可以使用单引号把多字短语组合为一个检索单位。单引号的另外一个作用是能够确保系统按文字解释一定的元字符，如“\$”符号等。



即使使用引号，命令历史机制中使用的元字符“!”总是作为元字符解释的，除非在前面加转义符号“\”。推而广之，如果想要让grep命令按照普通字符解释“&”、

“!”、“\$”、“?”、“.”、“;”和“\”等元字符，必须在每个元字符前面加转义符号“\”。

例如，如果输入下列命令，将会显示stray.bird文件中的所有文本行：

```
$ grep $ stray.bird
I cannot choose the best
The best choose me

Roots are the branches down in the earth
Branches are the roots in the air
$
```

然而，如果输入下列命令，则只会显示包含“\$”字符的文本行：

```
$ grep '\$' stray.bird
$
```

有关grep命令的更多内容，可以查阅系统提供的命令参考手册。

5.16 排序

排序是一种经常需要用到的工具。使用sort命令可对输入数据或文件内容进行排序，使其按照一定的顺序逐行显示。sort命令的语法格式简写如下：

```
sort [-bdfimnru] -k start [,stop] -t char -o outfile [file]
```

其中，file表示准备排序的文件。其他常用的命令选项及其功能如表5-5所示。

表5-5 sort命令的部分选项及其简单说明

选项	GNU选项	简单说明
-b		表示忽略文本行前置的空白字符
-d	--dictionary-order	表示仅考虑字母数字和空格字符，忽略标点符号等字符，按字典顺序排序
-f	--ignore-case	表示排序时忽略字母的大小写，所有的小写字母均做大写字母处理，统一进行排序
-k start[,stop]	--key=start[,stop]	表示排序字段的字段位置，或者排序字段的起止字符位置及范围。如果没有明确指定排序字段，默认的排序字段为整个文本行
-n	--numeric-sort	表示按照字符串的数值而不是文字进行排序
-o outfile	--output=outfile	用于指定存储排序结果的输出文件（默认值为标准输出）
-r	--reverse	表示按照从大到小或反向字符的顺序排序
-t char	--field-separator=char	用于指定除空白字符之外的其他字段分隔符

排序时，如果未指定排序字段，sort命令将会以整个文本行为单位，逐行比较参与排序的输入文件或数据。如果利用“-k”选项指定了文本行的排序字段，而未指定字段分隔符，sort命令将会以空格或制表符等空白字符作为分隔符，按照指定的排序字段对输入文件或数据进行排序。在指定排序字段时，start表示起始字段，stop（如果存在）表示结束字段。在指定起止

排序字段时，可以采用“f[c]”形式指定字段及其起始字符，其中f是字段的序号，选用的c是字段中的起始字符位置，字段序号和字符位置均从1开始编号。在指定起始字段时，如果忽略了起始字符c，则表示指定字段的第一个字符。在指定结束字段时，如果忽略了起始字符c，则表示指定字段的最后一个字符。假定namelist是一个内部通信录，下列sort命令表示以第2个自然字段作为排序字段，对指定文件的数据进行排序：

```
$ cat namelist
Sales      John Smith      2880
Sales      David Cooper    2886
Support    Frank Moore     2661
Support    Roger Kent      2662
Admin      Lucy Grant       2445
Admin      Alice Thomas    2449
$ sort -k 2 namelist
Admin      Alice Thomas    2449
Sales      David Cooper    2886
Support    Frank Moore     2661
Sales      John Smith      2880
Admin      Lucy Grant       2445
Support    Roger Kent      2662
$
```

下列sort命令表示从第4个自然字段的第2个字符开始，对指定文件的输入数据进行排序：

```
$ sort -k 4.2 namelist
Admin      Lucy Grant       2445
Admin      Alice Thomas    2449
Support    Frank Moore     2661
Support    Roger Kent      2662
Sales      John Smith      2880
Sales      David Cooper    2886
$
```

在使用sort命令排序时，可以同时指定多个排序字段，按照主次分类的形式进行排序。例如，下列sort命令表示以第2个和第3个自然字段作为排序字段，对指定文件的输入数据进行排序：

```
$ sort -k 1 -k 2 namelist
Admin      Alice Thomas    2449
Admin      Lucy Grant       2445
Sales      David Cooper    2886
Sales      John Smith      2880
Support    Frank Moore     2661
Support    Roger Kent      2662
$
```

此外，sort命令经常用于排序其他命令的输出结果。例如，“ls -l”命令的输出通常是按文件名的字符顺序打印文件列表的，为了按照文件的大小，从大到小排序，可以观察“ls -l”命令的输出，确定文件大小字段的位置（第五列），然后利用“-k”、“-r”（从大到小排序）和“-n”（按数值的大小排序）选项进行排序，示例如下：

```
$ cd /var/log
$ ls -l syslog*
-rw-r----- 1 syslog adm 1391559 2009-11-20 18:15 syslog
-rw-r----- 1 syslog adm 1121513 2009-11-20 10:29 syslog.1
-rw-r----- 1 syslog adm 151847 2009-11-19 01:32 syslog.2.gz
-rw-r----- 1 syslog adm 281149 2009-11-18 11:19 syslog.3.gz
$ ls -l /var/log/syslog* | sort -k 5 -rn
-rw-r----- 1 syslog adm 1391653 2009-11-20 18:17 syslog
-rw-r----- 1 syslog adm 1121513 2009-11-20 10:29 syslog.1
-rw-r----- 1 syslog adm 281149 2009-11-18 11:19 syslog.3.gz
-rw-r----- 1 syslog adm 151847 2009-11-19 01:32 syslog.2.gz
$
```


第6章 vim编辑器

在UNIX系统vi编辑器的基础上，Linux系统提供了克隆版的vim编辑器（参见<http://www.vim.org>）等。vim编辑器是对vi的扩充与增强，它提供了许多附加的功能特性，且与vi几乎是完全兼容的。vim可以运行在许多平台上，包括Windows、Macintosh、UNIX和Linux系统。利用vim，可以编辑文件，开发应用程序。

vim是一个极其强有力的文本编辑工具，如果能够掌握其规律，vim将是一个非常好的开发工具。但在使用vim编辑器时，用户需要记住编辑器当前所处的工作模式。即便如此，只要经过一定的实践，用户通常很快就会熟练地掌握vim，得心应手地处理文本文件。vim提供大量的命令，供用户编辑文件。本章将按照操作类型，介绍最基本的vim命令。

6.1 启动vim编辑器

6.1.1 创建文件

在Linux系统的命令提示符下输入下列命令，即可启动vim编辑器：

```
$ vim myfile
```

如果myfile文件存在，上述命令将会打开指定的文件，同时在编辑窗口中显示文件第一页的数据内容。如果指定的文件不存在，vim将会打开一个新文件，出现如图6-1所示的编辑窗口。



图6-1 vim编辑器界面

此时，光标将处于编辑窗口左上角的位置，屏幕左边的波浪符“~”表示空行，说明这是一个空文件。注意，启动vim时可以同时指定多个文件名参数，意味着同时编辑多个文件，也可以不指定文件名，等到完成文件编辑之后再使用“w”命令写入一个新文件，然后退出vim。

如果运行vim命令时未指定文件的名称，将会显示如图6-2所示的内容。此时，可以直接编写文件，等完成之后再使用“:w filename”命令保存写就的文件，然后使用“:q”命令退出vim。初学者也可以使用“:help”命令获取vim的帮助信息，如图6-3所示。

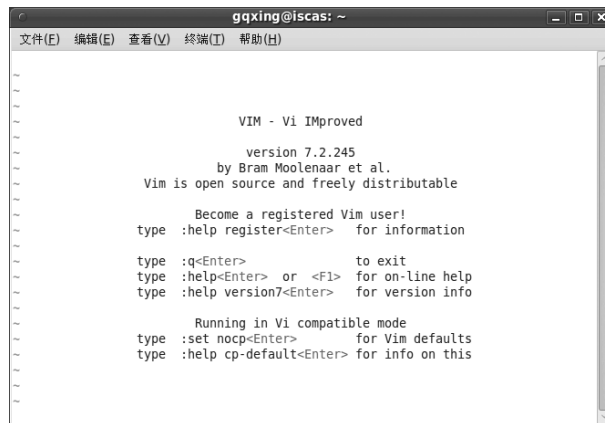


图6-2 vim编辑器初始界面

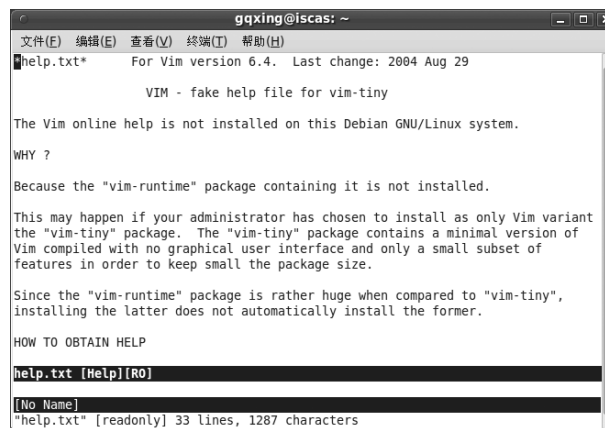


图6-3 vim编辑器帮助界面

6.1.2 状态行

编辑窗口的最后一行是vim的状态行，用于显示编辑器的状态、编辑过程中出现的出错信息、光标所在的行列位置、删除或复制的行数等。初始启动时，状态行将会显示文件的名字、文件的行数，以及文件的字节计数。在图6-1中，状态行表示打开的是一个新（空）文件。

6.2 vim编辑器的工作模式

vim分为命令和输入两种工作模式。任何时刻，vim编辑器总是处于命令和输入两种工作模式之一。当启动vim编辑器、打开或创建一个文件时，vim即处于命令模式。通过发布vim命令，可使vim处于输入模式。在输入模式下，可以输入数据，编写自己的应用程序。而在命令模式下，可以输入vim命令，执行特定的vim编辑功能。命令模式是vim的默认工作模式。通过使用vim命令和Esc键，可以在两种工作模式之间相互转换。

在某些情况下，vim并不提示编辑器当前所处的工作模式。对于vim的新用户来讲，区分命令模式与输入模式也许是最感困惑的问题。但只要记住一点，就可以做到心中有数。即无论何时，只要按下Esc键，不管vim当前处于何种工作模式，总会进入命令模式。因此，多敲几次Esc键是vim用户的常见动作。

第一次使用vim打开文件时，vim总是处于命令模式。在能够输入任何文本之前，首先必须输入vim的数据输入命令。例如，输入“i”（“插入”）字符命令，即可在当前光标所处字符位置之前插入数据。输入“a”（“附加”）字符命令，即可在当前光标所处字符位置之后附加数据。本章的后面将详细介绍各种数据输入命令。

无论何时需要返回命令模式，只需按下Esc键即可。如果不能确定vim当前处于何种工作模式，只要按下Esc键，即可确保vim总是处于命令模式，然后再决定下一步怎么做。如果在vim处于命令模式时按下Esc键，或按下其他不合法的键时，终端将会发出鸣叫，或发生屏幕闪烁现象，但不会影响正在编辑的文件。

6.2.1 输入模式

为了在前面打开的my file示例文件中输入数据，可输入vim的“插入”命令“i”。这一命令将使vim编辑器从命令模式转入输入模式。

现在，即可尝试输入几行数据，在每行数据输入结束后按下Enter键。在输入数据的过程中，可以利用退格键（Backspace）做简单的校正（在按下Enter键之前）。在整个输入过程结束之后，按下Esc键，即可返回命令模式。此时，光标将处于刚才输入的最后一个字符位置。然后，还可以利用各种vim命令，对输入的数据进行校正。

6.2.2 命令模式

如上所述，当利用vim打开一个文件时，vim将处于命令模式。在命令模式下，可以输入各种vim命令，以便完成各种编辑功能。vim命令几乎都是由一个字符、两个字符或一个选用的数字加字符组成的。通常，字母相同但大小写不同的字符命令，其意义相同，但作用则完全不同。例如，字符命令“a”意味着在当前光标所处字符位置之后附加数据，而“A”则意味着在当前光标所在行的行尾附加数据。

大多数vim命令不需要按Enter键即可立即执行。但是，以冒号“:”开始的命令需要在输入命令之后再按Enter键。在命令模式下输入冒号“:”时，冒号“:”将会出现在编辑窗口底部最后一行的左下角，然后即可接着输入编辑命令。

以冒号开始的命令实际上是ex命令，ex与vim命令是同一编辑程序的两个不同的用户界面，vim提供面向屏幕的用户界面，而ex则提供面向命令行的用户界面。所有的ex命令均可在vim中使用。在输入冒号之后，实际上已经切换到面向命令行的ex用户界面。这种切换方式使用户能够在不离开vim的情况下执行许多文件编辑命令，甚至也可以执行其他Shell命令（参见6.5节）。

6.3 保存文件与退出vim

在使用vim编辑文件期间，用户所做的任何编辑处理并未直接反映到实际的文件中。实际上，整个编辑过程将被保存到vim于内存中临时创建的一个文件副本中。仅当发布“w”（“写”）等命令时，内存缓冲区中的内容才能永久性地保存到磁盘上的文件中。

vim编辑器的这种处理方式既有积极的一面，也有丢失数据之忧。可取之处是在退出文件编辑时，用户可以放弃编辑期间所做的任何修改，而不影响原有的文件。缺点是当系统发生故障时，很有可能丢失内存缓冲区中保存的数据。

因此，最好的做法是注意随时保存数据，特别是当编辑重要的文件时。

vim 编辑器提供许多命令，用于把内存缓冲区中的数据内容保存到磁盘文件中，然后退出 vim 编辑器。这些命令允许用户选择“保存并退出”、“强制退出而不保存”等编辑器退出方式（如表6-1所示）。


表6-1 vim的保存文件和退出命令

命令	简单说明
:w	保存编辑后的文件内容，但不退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中
:w!	强制写文件，即强制覆盖原有的文件。如果原有文件的访问权限不允许写入文件，例如，原有的文件为只读文件，则可使用这个命令强制写入。但是，这种命令用法仅当用户是文件的属主时才适用，而超级用户则不受此限制
:wq	保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。另外一种替代的方法是用ZZ命令
:wq!	强制保存文件内容后退出vim编辑器。这个命令的作用是把内存缓冲区中的数据强制写到启动vim时指定的文件中，然后退出vim编辑器
ZZ	使用ZZ命令时，如果文件已经做过编辑处理，则把内存缓冲区中的数据写到启动vim时指定的文件中，然后退出vim编辑器。否则只是退出vim而已。注意，ZZ命令前面无需加冒号“:”，也无需按Enter键
:q	在未做任何编辑处理而准备退出vim时，可以使用此命令。如果已做过编辑处理，则vim不允许用户使用“:q”命令退出，同时还会输出下列警告信息： No write since last change (:quit! overrides)
:q!	强制退出vim编辑器，放弃编辑处理的结果。如果确实不需要保存修改后的文件内容，可输入“:q!”命令，强行退出vim编辑器
:w filename	把编辑处理后的结果写到指定的文件中保存
:w! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已经存在，则覆盖现有的文件
:wq! filename	把编辑处理后的结果强制保存到指定的文件中，如果文件已经存在，则覆盖现有的文件，并退出vim编辑器

6.4 vim编辑器的基本命令

本节将分类介绍vim提供的各种编辑命令:

- 移动光标位置;
- 输入文本数据;
- 修改和替换文本;
- 撤销先前执行的文本编辑命令;
- 删除文本;
- 重复执行先前的命令。

 **注意**

vim 命令是严格区分大小写字母的。即使命令形式完全相同，如果不注意区分大小写字母，将会产生完全不同的效果。

6.4.1 移动光标位置

当启动vim编辑器时，光标将处于编辑窗口左上角的位置，并处于命令模式。在命令模式下，可以使用表6-2所示的字符命令移动光标位置。

表6-2 光标移动命令

命令	简单说明
← ↑ ↓ →	方向箭头键。可在编辑窗口上将光标左移、上移、下移和右移一行或一个字符位置
h k j l	其功能与箭头键完全相同。在无法使用箭头键（如远程访问）时，可以使用这4个键分别替代相应的箭头键
-	把光标移至上一行的第一个起始字符位置（第一个非空白字符位置）
Enter键	把光标移至下一行的第一个起始字符位置（第一个非空白字符位置）
退格键	光标左移一个字符位置
空格键	光标右移一个字符位置
Ctrl-F	往下（文件结尾方向）滚动一屏。在命令方式下按Ctrl-F键，编辑窗口中将会显示文件下一页的内容，光标也同时移至下一页的左上角位置
Ctrl-B	往上（文件开始方向）滚动一屏。在命令方式下按Ctrl-B键，编辑窗口中将会显示文件前一页的内容，光标也同时移至前一页的左下角位置
Ctrl-U	往下滚动半屏
Ctrl-D	往上滚动半屏
Ctrl-E	编辑窗口中的文件内容整体上移一行
Ctrl-Y	编辑窗口中的文件内容整体下移一行
H	把光标移至编辑窗口顶部第一行的起始字符位置（第一个非空白字符位置）
M	把光标移至编辑窗口中间一行的起始字符位置（第一个非空白字符位置）
L	把光标移至编辑窗口底部最后一行的起始字符位置（第一个非空白字符位置）
w	光标右移一个字。如果相邻的两个字之间有标点符号，光标将移至标点符号位置
W	光标右移一个字。即使相邻的两个字之间有标点符号，也忽略之
b	光标左移一个字。如果相邻的两个字之间有标点符号，光标将移至标点符号位置
B	光标左移一个字。即使相邻的两个字之间有标点符号，也忽略之
e	把光标移至当前字（或下一个字）的最后一个字符位置
E	同上，只是以空格字符作为字的分隔符
^	把光标移至当前行的起始位置，即当前行的第一个非空白字符位置
0（零）	同上
\$	把光标移至当前行的行尾，即当前行的最后一个字符位置
[n]G	将光标移至指定行的行首位置。其中n表示行号，默认情况下转至文件的最后一行。因此，为了转到当前文件的最后一行，只需输入“G”命令即可。为了移至文件的第一行，可输入“1G”命令。实际上，利用“nG”命令，可以转至当前文件的任何一行。例如，为了修改文件的第60行，只需输入“60G”命令，即可立即跳转到第60行的行首位置

命令	简单说明
(把光标移至一个完整句子的句首位置
)	把光标移至一个完整句子的句尾位置
{	把光标移至一个完整段落的段首位置
}	把光标移至一个完整段落的段尾位置

6.4.2 输入文本

vim 提供许多命令，使用户能够输入文本数据。表6-3给出了vim 编辑器中最常用的几种数据输入命令。注意，这些命令将使vim 进入数据输入方式，同时在编辑窗口的左下角（即状态行左边）显示“— INSERT —”状态信息，表示vim 当前正处于输入模式。为了使用这些命令，首先必须确保vim 处于命令模式（多按几次Esc键总能保证vim 处于命令模式）。

表6-3 数据输入命令

命令	简单说明
a	使用“a”命令，可在光标当前所在字符位置之后输入数据，输入的数据数量不限，输入结束后可按Esc键退出输入模式。在发布“a”命令之前，可将光标移至任何文本位置
A	使用“A”命令，可在光标当前所在行的行尾（即最后一个字符之后）输入数据，输入的数据数量不限，直至按下Esc键。在发布“A”命令时，不管光标处于任何位置，都会移至当前文本行的行尾
i	使用“i”命令，可在光标当前所在字符位置之前输入数据，输入的数据数量不限，输入结束后可按Esc键返回命令模式。在发布“i”命令之前，可将光标移至任何文本位置
I	使用“I”命令，可在光标当前所在行的行首（即第一个非空白的起始字符之前）输入数据，输入的数据数量不限，直至按下Esc键。在发布“I”命令时，不管光标处于任何位置，都会移至当前文本行的行首
o	使用“o”命令，可在光标当前所在行之后插入数据，行数不限，直至按下Esc键结束
O	使用“O”命令，可在光标当前所在行之前插入数据，行数不限，直至按下Esc键结束

6.4.3 修改与替换文本

为了修改或替换文本数据，vim 提供若干不同的命令，用于校正文本数据。用户可以根据具体情况，灵活地予以选用（如表6-4所示）。在表6-4所列的编辑命令中，除了“r”和“~”命令不显示任何信息，“R”命令将会在编辑窗口的左下角显示“— REPLACE —”状态信息，其他命令均显示“— INSERT —”状态信息，表示vim 当前正处于输入模式。

表6-4 修改与替换命令

命令	简单说明
C	替换部分文本行。从光标位置开始直至行尾，替换其间的所有数据，直至按下Esc键结束
cw	替换单个字。为了替换一个整字，可将光标移至单字的字首位置，然后输入“cw”命令，接着输入新的文字。输入的字符数量不限，输入结束后可按Esc键返回命令模式。为了修改单字（后面部分）的一部分，可将光标移至该字欲保留部分右边的字符位置，输入“cw”命令后即可校正该字，结束后按Esc键返回命令模式

(续表)

命令	简单说明
[n]cc	替换整个文本行。为了替换一整行文本，只需把光标移至目标行的任何字符位置，然后输入“cc”命令。此时，当前文本行将会消失，留下一个空行位置，等待用户输入新的数据，输入的数据数量或行数不限。输入结束后，按Esc键返回命令模式。为了替换多行文本，可把光标移至目标行的第一行，然后输入“ncc”命令，其中n表示需要替换的文本行的数量
[n]s	替换单个字符。为了替换光标位置的单个字符，可输入“s”命令，之后可以输入任何数量或行数的数据。输入结束后，按Esc键返回命令模式。为了替换从光标位置开始的多个字符，可输入“ns”命令，其中n表示需要替换的字符数量
S	替换文本行。为了替换光标当前所在的文本行，可输入“S”命令，之后可以输入任何数量或行数的数据。输入结束后，按Esc键返回命令模式
r	替换单个字符。使用“r”命令，可用随后输入的字符替换光标位置的单个字符。与“s”命令不同，“r”命令只能替换一个字符，替换后，vi编辑器自动返回命令模式（不需要再按Esc键）
R	替换多个字符。使用“R”命令，可以从光标位置开始替换多个字符，数量不限，直至按下Esc键结束
[n]~	转换光标位置字母的大小写。如果在输入“~”之前输入一个数字，可以一次转换多个字母的大小写。此外，如果一直接住波浪号“~”键，能够连续转换多个字母的大小写。

6.4.4 撤销先前的修改

在编辑文本期间，以及准备把加工后的数据保存到文件之前，有时可能需要放弃某些编辑处理的结果。vim的“u”和“U”命令能够撤销先前执行的编辑命令的处理结果，使vim回退到先前的处理状态。然后，可从先前的处理位置开始继续进行编辑加工。撤销命令及其说明如表6-5所示。

表6-5 撤销命令

命令	简单说明
u	用于撤销先前执行的编辑命令。如果在编辑过程中出现失误，或编辑后又改变了决定，可以使用vim的“u”命令，撤销先前执行的编辑命令。注意，输入“u”命令后不需要按Esc键。连续输入“u”命令后能够以回溯的方式，依次撤销先前执行的所有编辑命令
U	撤销或恢复对当前文本行所做的全部编辑处理。使用vim的“U”命令可以撤销或恢复最近一次编辑处理的文本行。也就是说，“U”命令仅适用于最近一次修改的文本行。同样，输入“U”命令后也不需要按Esc键

6.4.5 删除文本

利用表6-6给出的vim命令，可以删除指定的字符、字或文本行数据。

表6-6 删除命令

命令	简单说明
[n]x	删除字符。为了删除单个字符，可将光标移至准备删除的字符位置，然后输入“x”命令即可。“x”命令除删除指定的字符之外，还将删除字符占用的空间位置——当从某个单字中间删除一个字符时，余下的字符将合并为一个新的字，中间不会留下任何间隙。利用“x”命令，也可以删除文本行中的空格字符。为了删除多个字符，可将光标移至准备删除的字符串起始位置，然后输入“nx”命令，其中的n表示字符的数量

命令	简单说明
[n]X	删除字符。为了删除光标当前所在位置的前一个字符，可以使用大写的“X”命令。为了删除多个字符，可将光标移至准备删除的字符串的右边，然后输入“nX”命令，其中的n表示字符的数量
dw	删除单个字或部分字。为了删除一个整字，可以把光标移至该字的起始字符位置，然后输入“dw”命令。相应的字及其占用的空间位置将一并删除。为了删除单字的右边部分，可将光标移至该字欲保留部分的后面，输入“dw”命令，即可删除单字的右边部分
[n]dd	删除文本行。为了删除整个文本行，可以把光标移至文本行的任何位置，然后输入“dd”命令，整个文本行及其占用的空间将会一并删除。为了同时删除多个文本行，可将光标移至准备删除的第一个文本行，然后输入“n dd”命令，其中的n表示准备删除的行数（包括当前行在内）
D	删除文本行的行尾部分。为了删除文本行右边的部分文字，可将光标移至文本行欲保留部分的后面，输入“D”命令，即可删除文本行的行尾部分

6.4.6 复制、删除与粘贴文本

许多字处理软件都提供“复制-粘贴”与“剪切-粘贴”的文本行处理方式。vim编辑器也提供这样的功能。在vim编辑器中，与“复制-粘贴”等价的处理过程是先用“yy”命令复制文本行，接着再用“p（或P）”命令实现文本行的实际复制；与“剪切-粘贴”等价的处理过程是先用“dd”命令删除文本行，接着再用“p（或P）”命令实现文本行的移动。在上述两种组合方式的基础上，如果在“yy”或“dd”命令之前再输入适当的数字，还可以实现一组文本行的复制和移动处理（复制、删除与粘贴命令如表6-7所示）。

表6-7 复制、删除与粘贴命令

命令	简单说明
[n]yy	复制文本行。实际上，“yy”命令只是把文本行的数据内容保存到粘贴板中。一个复制动作需要同时使用两个命令才能完成：即“yy”与“p（或P）”命令。 因此，为了复制一个文本行，可按下列步骤执行： （1）把光标移至准备复制的文本行的任何位置； （2）输入“yy”命令； （3）再把光标移至目标文本行的任何位置； （4）输入“p”命令，将粘贴板中的数据内容复制到光标所在行的下面； （5）或输入“P”命令，将粘贴板中的数据内容复制到光标所在行的上面。 如果在输入“yy”命令之前输入数字，则可以同时复制多个文本行。例如，如果想要复制10行数据，可输入“10yy”命令，这将从光标当前所在行开始的10行数据复制到粘贴板中。此时，vim编辑器将会在编辑窗口底部显示一条信息：“10 lines yanked.”，表示命令已成功地执行
[n]Y	其功能等同于“yy”命令
[n]dd	删除文本行。为了把一个或若干文本行移至某个位置，需要先删除文本行，然后再粘贴到适当的位置，因此也需要同时使用两个命令才能完成：即“dd”与“p（或P）”命令。 例如，为了移动5行数据，可以把光标移至欲删除文本行的任何位置，发布“5dd”命令，然后把光标移至插入位置，接着输入“p（或P）”命令，即可把文本行移至当前文本行的下方（或上方）
p（小写）	把粘贴板中的文本数据复制到光标所在文本行的下面
P（大写）	把粘贴板中的文本数据复制到光标所在文本行的上面

6.4.7 重复执行命令

许多vim命令前面都可以加一个计数值，表明相应的命令需要重复执行的次数。在前面的讨论过程中，我们实际上已经用到了这样的命令。例如，“3dd”命令意味着删除文本行的动作需要执行三次，最终结果是删除三行数据。“2dw”命令意味着删除两个字，“4x”命令意味着删除4个字符（包括空格）。

另外，也可以使用计数命令方式移动光标。例如，“3w”命令意味右移三个字，2Ctrl-F意味往前滚动两屏。

使用句点“.”命令也可以重复执行先前的文本编辑命令。例如，如果用户刚刚使用“dd”命令删除了一个或多个文本行，此时可以把光标移至准备删除的文本行中，仅仅输入一个句点“.”命令即可重复执行删除文本行的处理动作。对于复杂的编辑命令，句点命令尤其方便。

6.5 使用ex命令

事实上，在需要处理大块文本行的情况下，与上述的“复制-粘贴”与“剪切-粘贴”处理方式相比，利用ex命令还可以实现更精确、更方便的编辑处理。使用ex命令时，无需在编辑窗口中计数文本行的数量，然后再寻找插入点。用户只需提供一个准备复制或移动的文本行的范围，然后指定插入点的行号即可（当然，删除时无需指定插入点）。注意，在输入ex命令时，首先需要输入一个前置的冒号“:”。

6.5.1 显示行号

使用ex命令时，通常需要知道文本行的编号。为了在编辑的文件中显示行号，可输入下列命令：

```
:set nu
```

按下Enter键后，新加的行号将会出现在编辑窗口的左边。注意，这些行号实际上并不存在于文件中，只是为方便用户的编辑处理而出现在编辑窗口中，以增加文本数据的可读性（如图6-4所示）。

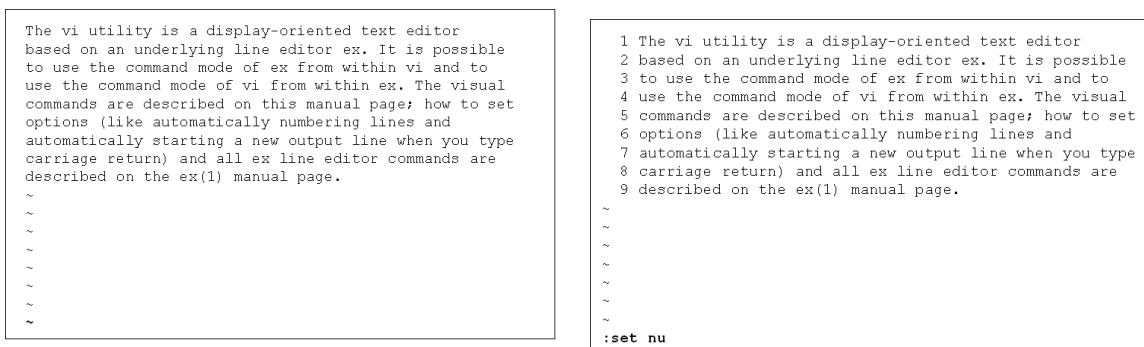


图6-4 显示行号

若想关闭行号显示，可输入下列命令。

```
:set nonu
```

另外，为了随时了解当前文本行的行号位置，可在命令模式中按下Ctrl-G组合键。vim编辑器将会在编辑窗口的底部显示当前行的行号位置，以及当前文件的名称等信息。

6.5.2 多行复制

ex复制命令的基本语法格式如下：

```
:line#1,line#2 co line#3
```

其中，前两个数字（即line#1和line#2，中间以逗号分开）用于指定需要复制的文本行的范围，第三个数字（即line#3）表示插入点的行号。例如，为了把my file文件的第1行至第5行复制到当前文件的第12行之后，可以使用下列命令：

```
:1,5 co 12
```

在指定文本行的范围时，可以使用下列缩写形式：

- 句点 “.”：表示当前行，意味着从当前行开始。
- 美元符号 “\$”：表示文件的结尾，即文件的最后一行。

因此，为了把当前行至后续第5行复制到第12行之后，可以使用下列命令：

```
:. ,5 co 12
```

为了把第6行直至文件最后一行复制到第2行之后，可以使用下列命令：

```
:6,$ co 2
```

6.5.3 移动文本行

ex移动命令的基本语法格式类似于复制命令：

```
:line#1,line#2 m line#3
```

在移动文本行时，可以采用与复制文本行相同的方式指定文本行的范围和插入点，包括使用缩写的句点 “.” 和美元符号 “\$”。两者的差别仅在于“移动”命令将会把指定范围的文本行从一个位置整块地照搬到另一个指定的位置。

例如，为了把第1行至第5行移至第12行之后，可以使用下列命令：

```
:1,5 m 12
```

6.5.4 删除文本行

删除文本行时，也可以采用相同的方式指定文本行的范围，包括使用缩写的句点 “.” 和美元符号 “\$”。为了删除多个连续的文本行，可以使用下列命令形式实现：

```
:line#1,line#2 d
```

例如，为了删除文件中的第1行至第5行，可输入下列命令：

```
:1,5 d
```

6.6 检索与替换

vim 提供若干命令，能够以检索指定字符串的方式，直接跳转至期望的文件位置。另外，vim 还提供强有力的全局检索与替换功能。

6.6.1 字符串检索

字符串是由一个或多个连续的字符组成的。字符串可以包含字母、数字、标点符号、特殊字符、空格、制表符或回车字符。字符串既可以是一个语法意义上的单词，也可以是单词的一部分。为了检索字符串，可以使用如表6-8所示的命令。

表6-8 检索命令

命令	简单说明
<code>:/str</code>	检索给定的字符串。vim 将从当前光标位置开始检索，当找到指定的字符串时，光标将移至第一个出现的字符串位置。例如，为了检索meta，可以输入 <code>/meta</code> 命令，然后按Enter键
<code>:?str</code>	从当前光标位置开始，反向检索给定的字符串
<code>n</code>	从当前光标位置开始，继续检索下一个匹配的字符串
<code>N</code>	从当前光标位置开始，反向检索下一个匹配的字符串
<code>/</code>	同“n”命令，从当前光标位置开始，重复执行先前的检索，但在输入“/”字符之后还需要按Enter键
<code>?</code>	同“N”命令，从当前光标位置开始，反向重复执行先前的检索，但在输入“?”字符之后还需要按Enter键
<code>:/pat/+n</code>	将光标移至匹配的字符串“pat”所在文本行之后的第n行
<code>:?pat?-n</code>	将光标移至匹配的字符串“pat”所在文本行之前的第n行

为了检索字符串，可在“:/”后面附加准备检索的字符串，然后按Enter键。vim 将会从当前光标位置开始向下（文件结尾方向）检索。如果发现匹配的字符串，vim 将会把光标移至检索方向第一个出现的字符串位置。

例如，为了检索字符串“utility”，可输入 `:/utility` 命令，然后按Enter键。如果接着输入“n”命令，可以继续检索下一个匹配的字符串。如果输入大写的“N”命令，则可以逆向检索前一个匹配的字符串。

为了在当前编辑的文件中向前（文件开始方向）逆向检索，可以使用“:?”代替“:/”命令。在任何情况下，“n”和“N”的检索方向仍然保持不变。但从逻辑上讲，则恰好与“:?”的检索方向相反。

通常，vim 的字符串检索是严格区分大小写字母的。例如，在检索“china”时，vim 不会发现“China”。如果想在检索期间忽略大小写的差异，可以输入 `:set ic` 命令。检索完成后，为了返回默认的匹配方式（区分字母大小写），可输入 `:set noic` 命令。

如果发现检索的字符串，vim 将会把光标移至（并停留在）目标字符串的第一个字符位置。如果未发现检索的字符串，vim 将会在编辑窗口的底部（状态行中）输出“Pattern not found”信息，说明检索失败。

在检索过程中，某些特殊字符（“/”、“&”、“!”、“.”、“'”、“*”、“\$”、“\”和“?”）具有特定的意义，如果检索字符串中本身也包含这样的字符，必须在其前面增加转义符号“\”，使vim能够按普通字符处理。例如，为了检索字符串“anything?”，可输入“:./anything\?”命令。为了检索一个本身就包含转义符号“\”的字符串，可在转义符号之前再加一个转义符号，即输入两个反斜线“\\”。

6.6.2 模式检索

- 利用vim提供的模式检索命令（如表6-9所示），还可以使字符串的检索更精确、更有效。
- 仅检索出现在行首位置的字符串;
 - 仅检索出现在行尾位置的字符串;
 - 仅检索出现在字首位置的字符串;
 - 仅检索出现在字尾位置的字符串。

表6-9 模式检索命令

命令	简单说明
:/^search	为了检索仅仅出现在行首位置的字符串，可以在字符串前面冠以一个上箭头“^”字符，使vim仅仅匹配行首位置，而忽略出现在其他位置的字符串。例如，为了从当前行开始，检索以“From”为起始字符串的文本行，可以输入“/^From”命令。如果找到匹配的字符串，光标将会移至目的文本行的行首位置。如果想直接定位匹配的字符串，可接着输入“n”命令。如果找不到匹配的字符串，vim将会在编辑窗口底部显示“Pattern not found”信息，而光标则继续停留在原来的位置
:/search\$	为了检索仅仅出现在行尾位置的字符串，可以在字符串后面附加一个“\$”字符，使vim仅仅匹配行尾位置，而忽略出现在其他位置的字符串。例如，为了从当前行开始，检索以“end”字符串结尾的文本行，可以输入“/end\$”命令。同样，如果找到匹配的字符串，光标将会移至目的文本行的行首位置。如果想直接定位匹配的字符串，可接着输入“n”命令。如果找不到匹配的字符串，vim将会在编辑窗口底部显示“Pattern not found”信息，而光标则继续停留在原来的位置
:/\<search\>	为了匹配某个单词起始部分的字符串，可在检索字符串的前面冠以“\<”。为了匹配一个单词的结尾部分，可在检索字符串的后面附加“\>”。因此，为了准确地匹配一个完整的字，而非部分字符串，可在检索字符串前后加上“\<”和“\>”。例如，如果想在整个文件中检索“search”这一英文单词，可以输入“:/\<search\>”命令

此外，也可以在检索模式中使用“.”、“*”和“[...]”等通配符，使得检索更灵活。为了匹配任何一个字符，可以在检索模式的相应位置使用句点通配符“.”。例如，为了检索“fun”或“gun”，可以输入“:./un”命令。为了指定一个匹配范围，也可以使用范围通配符进行检索。例如，可以使用“:./[a-z]string”命令，使第一个字符仅限于小写字母。另外，也可以使用列举方式匹配限定的几个字符。例如，“:./[dm]string”命令表示仅检索以“m”或“d”为起始字符的字符串“string”，即仅检索“dstring”或“mstring”。为了检索以某个字符范围开头，中间含有某种模式的字符串，可以组合使用多个通配符进行检索。例如，为了检索以小写字母开头，中间含有“string”的字符串，可以使用“:./[a-z]*string*”命令。

6.6.3 字符串替换

字符串替换是在前述字符串检索的基础上实现的。因此，在字符串检索与替换的过程中可以使用任何通配符。

字符串替换命令的基本语法格式如下：

```
: [g] /search-string/s//replace-string/ [g] [c]
```

其中，第一个字符命令“g”表示全文检索，“s”表示替换，第二个字符命令“g”表示替换匹配的所有字符串，“c”表示在替换之前需经用户确认。因此，为了把文件中的所有字符串“BankA”替换为“BankB”，可输入下列命令：

```
:g/BankA/s//BankB/g
```

执行上述命令后，vim 将会把自当前行开始，直至文件结尾范围内的所有“BankA”全部替换为“BankB”。为了能够在处理之前先经用户确认，然后再替换，可以增加“c”命令，使vim 在执行每个替换之前请用户予以确认。例如，采用下列命令，可以使vim 在用“BankB”替换“BankA”之前，先请用户予以确认。输入“y”表示同意替换，输入“n”表示不同意替换。

```
:g/BankA/s//BankB/gc
```



利用Ctrl-C组合键，可以在中途停止这种“确认-替换”的交互式字符串替换方式。

6.7 编辑多个文件

6.7.1 编辑多个文件

vim 允许用户同时编辑多个文件。例如，为了在编辑file1文件的同时，也编辑file2文件，可以使用下列命令打开两个文件：

```
vim file1 file2
```

此时，vim 首先显示第一个文件file1，因此可以先行编辑file1。编辑结束后输入“:w”命令，保存file1文件。为了编辑file2，可以输入“:n”或“:n file2”命令。编辑结束后输入“:w”命令，保存file2文件。编辑结束后，可以输入“:q”命令，退出vim编辑器，或使用“:q!”命令，中途强制退出vim编辑器。

在编辑多个文件期间，可以随时使用“:e filename”或“:n filename”命令，直接转到指定的文件。也可以使用“:n”命令转到下一个文件，或使用“:n #”命令交替编辑最近处理过的两个文件。如果对文件做过任何编辑加工，在跳转之前，vim 将会要求用户使用“:w”命令保存当前文件，除非设置了vim的autowrite选项，vim 会在跳转之前自动保存修改后的文件。如果想放弃当前的修改，可以使用“:e! filename”命令，强行转至指定的文件，或使用“:q!”命令，强制退出vim编辑器。

在编辑过程中，使用vim的“r”命令能够很方便地把指定的文件读入（插入）当前文件的光标位置。这个命令的一般语法格式如下：

如果未指定行号, vim 将在光标当前所在文本行的后面插入文件。例如, 如果想在文件的第10行之后插入文件chap2, 可以输入下列命令:

另外，也可以先把光标移至第10行的位置，然后输入下列命令（这有助于确认插入位置的正确与否）：

为了把相邻的两个文本行合并为一行，可以把光标移至第一行，然后输入J命令。

6.8.1 临时设定vim运行环境

“set all” 将会输出vim编辑器当前支持的所有选项及其默认设置，如图6-5所示。



图6-5 vim编辑器当前使用的选项及其默认设置

```
:set option
```

```
:set option
```

其中，option是vim编辑器支持的选项名称。为了取消某个编辑器选项的设置，可以在选项前增加no字样：

```
:set nooption
```

表6-10 给出了vim编辑器支持的部分重要选项及其说明。

表6-10 vim编辑器支持的部分选项

选项	缩写	默认值	简单说明
all	无	无	在编辑窗口中列出编辑器支持的所有选项
magic	无	magic	在检索字符串时，下列字符通常具有特殊的意义： <ul style="list-style-type: none">· “.” 表示匹配任何一个字符· “[...]” 表示匹配指定字符集合或字符范围内的任何一个字符· “*” 表示匹配任何字符或字符串 在设置了nomagic选项之后，上述字符将会失去其特殊的意义。使用magic选项可以恢复其原有的特殊意义。注意，“^”（表示行首位置）和“\$”（表示行尾位置）总是具有特殊的意义，不受此选项设置的影响
autoindent	ai	noai	这个选项应与shiftwidth选项一起发挥作用，使新输入的文本行与上一行的起始位置自动对齐。当vim处于输入模式时，按下制表键Tab将会使光标移至下一个制表位置，按下Enter键将会使光标移至下一行，并与上一行的第一个字符位置对齐，按下Ctrl-T组合键将会使当前行移至下一个制表位置，而Ctrl-D组合键将会使当前行反向移至前一个制表位置。这一选项比较适用于程序员
autowrite	aw	noaw	在编辑多个文件的情况下，如果设置了autowrite选项，当使用“:e”或“:n”命令在文件之间切换时，vim将会在切换之前自动保存对当前文件所做的编辑处理。否则，vim将会提示用户把缓冲区中的内容写到磁盘文件中（如果当前编辑的文件中发生了任何变动）
ignorecase	ic	noic	在vim编辑器中，字符或字符串的匹配检索通常是严格区分大小写字母的。如果设置了这一选项，可以使vim在匹配检索过程中忽略大小写字母的差别
list	无	nolist	通常，vim将会按照正常的方式显示文本文件，如按照要求把制表符扩展为适当数量的空白字符，不显示回车字符等。如果设置了这个选项，vim将会把制表符显示为“^I”，在每一个文本行的后部附加一个美元符号“\$”等
laststatus	ls	ls=1	这个选项用于确定是否在编辑窗口中显示状态行。状态行中包括当前文件的名称、加号“+”标志（如果编辑的文件发生变动后尚未写到磁盘上）和光标的位置等。这个选项的有效值是0、1和2。其中0表示关闭状态行的显示；1表示仅当存在两个以上的文件编辑窗口时才显示状态行；2表示总是显示状态行
number	nu	nonu	在编辑文本文件时，vim编辑器通常不会显示文本行的行号。为了在每个文本行前面增加一个行号，可以设置这个选项。注意，vim编辑器显示的行号并非文件中的一部分，也不会存储到文件中，只是为了编辑的方便，提供一种临时的标记而已
readonly	无	noreadonly	对正在编辑的文件启用写保护机制，当编辑文件时，vim将会向用户发出警告提示，以避免修改或损害重要的文件

(续表)

选项	缩写	默认值	简单说明
report	无	report=2	这个选项使vim能够确定在删除或复制多少文本行时需要在状态行中显示影响的文本行信息，如“8 lines deleted”。当删除或复制较少的文本行（少于、等于指定值）时，vim不会显示任何信息。这个选项的默认值为2
scroll	scr	scr=nn	这个选项用于控制按下Ctrl-D组合键或Ctrl-U组合键时，前滚或后滚多少文本行。这个选项的默认值为编辑器窗口高度（窗口行数）的一半。为了修改Ctrl-D组合键或Ctrl-U组合键滚动的行数，通常有两种实现方式：一是在按下Ctrl-D组合键或Ctrl-U组合键之前首先输入一个数字；二是使用这个选项事先设定一个数值，如“:set scroll=10”
shell	sh	sh=path	在使用vim编辑器时，用户可以临时或长久地调用一个Shell，运行单个Linux命令，或交互地访问Shell，运行多个Linux命令。这个选项用于确定vim调用哪一个Shell。默认情况下，vim把这个选项设置为用户的注册Shell。为了选用不同的Shell，可以采用“:set sh=path”形式定义Shell，其中path为Shell的绝对路径名
shiftwidth	sw	sw=8	这个选项用于设定制表符的跳转位置，以便在输入模式下，当按下制表键Tab、Ctrl-T组合键或Ctrl-D组合键时光标能够自动地跳转到下一个或前一个制表符位置
showmatch	sm	nosm	输入右圆括号、花括号或方括号时提示相应的左圆括号、花括号或方括号。此选项对输入数学表达式或编写程序用到圆括号或花括号时比较有用
showmode	smd	smd	根据vim编辑器当前所处的工作模式，在编辑窗口左下角显示输入模式“-- INPUT --”，以及替换模式“-- REPLACE --”等信息
tabstop=8	ts	ts=8	设置制表键Tab的右移距离。默认值为8个空格
wrap	无	wrap	wrap选项用于控制vim编辑器怎样显示较长的文本行。为了使vim能够把较长的文本行延续到下一行，可以利用这个选项实现自动转行。如果设置了nowrap，vim将会截断较长文本行后部的超长部分（只是不显示，并非真正截掉超长部分）。这个选项的默认值为wrap
wrapmargin	wm	wm=0	指定编辑窗口的右边距。假定终端窗口为80列，如果使用“:set wm=8”命令设置右边距，则文本行的逻辑长度不能超出第72列的位置。当输入的文本行到达指定的边界时，vim编辑器将会在最接近边界的字（以空格字符为分界符）右边插入一个换行字符。这个选项可以帮助用户不用输入每一行之后再按Enter键。数值0（默认值）表示关闭这一功能，其他非0数值表示启用并设定编辑窗口的右边距
wrapscan	ws	wrapscan	这个选项影响字符串的检索方式。如果设置了ws选项，当检索到达文件的结尾仍未发现匹配的字符串时，vim将会从头开始继续检索。否则，在到达文件的结尾时，即使仍未发现匹配的字符串，vim也会停止检索
compatible	cp	cp	除非vimrc文件存在，默认情况下vim将会尝试采用与vi兼容的工作方式

另外，为了加快数据输入的速度，还可以使用下列命令形式，定义用户自己常用的缩写形式：


```
:ab abbr string
```

其中, `abbr`为`string`的缩写形式, `string`为实际上应出现在编辑器中的数据。例如, 如果使用下列命令:

```
:ab iscas 中国科学院软件研究所
```

则每当输入一个`iscas`单词时, vim编辑器将会自动代以“中国科学院软件研究所”字符串。

6.8.2 永久定制vim运行环境

上面介绍的方法只能临时地设置vim编辑器的当前运行环境, 一旦退出vim, 这种临时设置也随之作废。为了永久性地设置各种选项, 以免每次进入vim时都要重新设置, 可以在`profile`文件中设置`VIMINIT`变量, 其语法格式如下:

```
export VIMINIT='set arg1 arg2 .....
```

例如, 为了总是显示状态行, 在匹配检索过程中忽略大小写字母的差别, 可以把`VIMINIT`变量设置如下:

```
VIMINIT='set laststatus=2 ignorecase'; export VIMINIT
```

另外, 也可以把常用的vim选项及其定义(也即`set`命令可以设置的选项和定义)加到系统范围的初始化文件`/etc/vim/vimrc`中, 或用户主目录下的`vimrc`(或`exrc`)文件中。如此, 则每当调用vim编辑器时, vim都会自动读取此文件, 使定制的选项成为vim的永久性运行环境。下面是一个`vimrc`文件示例(注意, 此时`ex`命令之前无需加冒号):

```
set showmode
ab abc 中国农业银行
ab boc 中国银行
ab cbc 中国建设银行
ab icbc 中国工商银行
.....
```

实际上, 还可以在每个目录中创建一个自己的`vimrc`文件, 以适应不同的需要。例如, 可以在C++源程序所在的目录中创建一个适应于程序开发的`vimrc`文件, 可以在编写文档的目录中创建一个适应于文档编写的`vimrc`文件。但应注意, 仅当用户主目录的`exrc`文件包含“`set exrc`”命令时, 其他目录中的`vimrc`文件才能发挥作用。

6.9 其他说明

6.9.1 删除或替换特殊字符

在编辑文件时, 有时会遇到文件中含有不可打印的特殊字符的情况。例如, 由于Windows与Linux系统使用的行终止符不同, 对于Linux系统而言, 取自DOS或Windows系统中的文本文件会包含多余的回车字符(如图6-6所示)。

为了替换或删除这种特殊字符, 可以在vim的删除或替换命令中, 利用“`Ctrl-V`”键输入特殊字符的ASCII编码, 以删除或替换特殊字符。例如, 回车字符的ASCII编码为13, 对应于`Ctrl-`

M组合键。因此，可以采用下列替换命令，删除多余的“^M”字符。

```
:1,$ s/^M//
```

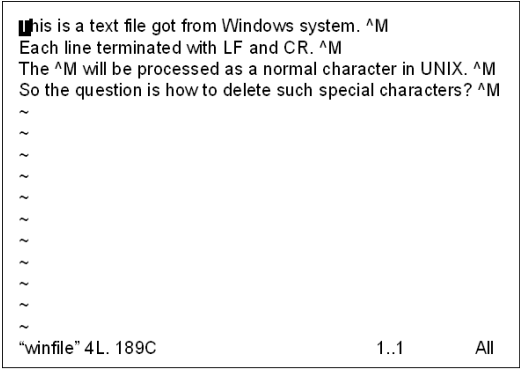



图6-6 DOS/Windows文本文件



注意

上述命令中的“^M”字符不能直接输入，必须先按Ctrl-V组合键，接着再按Ctrl-M组合键，才能输入“^M”字符（如图6-7所示）。

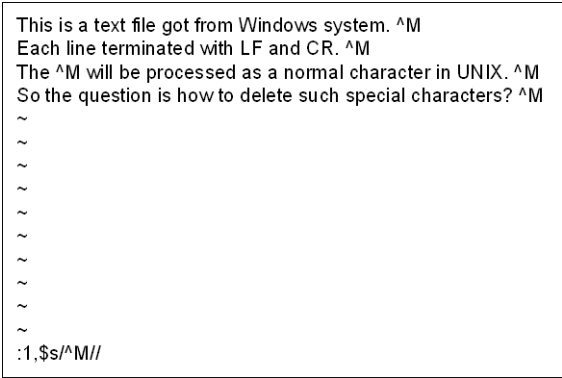


图6-7 替换回车字符

6.9.2 在编辑期间运行Linux命令

有时，我们可能需要在编辑过程中运行Linux命令，或需要把某个命令的输出作为文件的一部分，这时可以利用表6-11列出的命令，临时或长时间地运行其他Linux命令，或把Linux命令的运行结果引入正在编辑的文件中。

表6-11 辅助编辑命令

命令	简单说明
:sh	在编辑文件期间，如果想长时间地运行Shell命令，然后再返回vim编辑器，可输入“:sh”命令。此时，用户将处于正常的Shell命令行工作方式，当使用Ctrl-D组合键或exit命令退出Shell时，即可返回原来的vim编辑器，继续进行其他编辑处理
!:command	在编辑文件期间，如果想临时地运行某一个Shell命令，然后继续进行编辑处理，可输入“!:command”命令。此时，vim将会在不退出编辑器的情况下，执行指定的Shell命令。在Shell命令运行结束之后，按下Enter键，即可恢复原来的编辑处理状态

(续表)

命令	简单说明
!!command	在编辑文件期间，如果想把某个Shell命令的运行结果直接加到当前编辑的文件中，使命令的输出替代当前行，然后继续进行编辑处理，可输入“!!command”命令

6.10 vim编辑器命令总结

表6-12 分类列出了vim编辑器的基本命令及其简要说明。

表6-12 编辑器命令一览表

命令	简单说明
启动vim编辑器	
vim filename	打开原有的文件或创建一个新文件
vim	打开一个新文件，在编辑过程中或结束编辑时再指定文件名
vim -r filename	恢复因意外停机或终端连接中断而未及时保存最终编辑结果的文件
view filename	以只读方式打开文件。除了需要强制把编辑处理的最终结果写入文件进行保存之外，view的所有编辑功能均与vim无异
光标定位命令	
← ↑ ↓ →	将光标左移、上移、下移或右移一个字符（行）位置
h j k l	同上
-	光标上移一行
Enter键（或加号“+”）	光标下移一行
退格键	将光标左移一个字符位置
空格键	将光标右移一个字符位置（命令模式）
Ctrl-F	往下（文件结尾方向）滚动一屏
Ctrl-B	往上（文件开始方向）滚动一屏
Ctrl-D	往下滚动半屏
Ctrl-U	往上滚动半屏
Ctrl-E	编辑窗口中的文件内容整体上移一行
Ctrl-Y	编辑窗口中的文件内容整体下移一行
w	将光标右移一个字。光标停留在下一个字的字首位置
W	将光标右移一个字。光标停留在下一个字的字首位置（即使两个字之间存在标点符号）
b	将光标左移一个字。光标停留在下一个字的字首位置
B	将光标左移一个字。光标停留在下一个字的字首位置（即使两个字之间存在标点符号）
e	把光标移至当前字（或下一个字）的最后一个字符位置
E	同上，只是以空格字符作为字的分隔符

命令	简单说明
^	把光标移至当前文本行的起始位置，也即当前文本行的第一个非空白字符位置
0（零）	同上
\$	把光标移至当前文本行的行尾，也即当前文本行的最后一个字符位置
H	把光标移至编辑窗口顶部第一行的行首位置
M	把光标移至编辑窗口中间一行的行首位置
L	把光标移至编辑窗口底部最后一行的行首位置
插入文本数据	
a	在光标当前字符位置的后面输入文本数据
A	在光标当前所在文本行的行尾（也即最后一个字符位置）后面输入文本数据
i	在光标当前字符位置的前面输入文本数据
I	在光标当前所在文本行的行首（也即在第一个非空白的起始字符）前面输入文本数据
o	在光标当前所在文本行下面的行首位置输入文本数据
C	在光标当前所在文本行上面的行首位置输入文本数据
修改文本	
C	替换当前文本行光标所在字符位置之后的所有数据，以Esc键结束
cw	替换光标当前所在字符位置及之后的整个字或部分字，以Esc键结束
[n]cc	替换当前行，或从当前行开始的n行文本，以Esc键结束
[n]s	替换光标当前所在位置的单个字符，或从光标当前位置开始的n个字符，以Esc键结束
S	替换当前行，以Esc键结束
r	替换光标当前位置的单个字符
r Enter	断行。也可使用“a”或“i”命令加Enter及Esc键实现
R	从光标当前所在的字符位置开始，替换随后的所有字符，直至按下Esc键
xp	交换字符位置。交换当前光标位置开始的两个字符的位置
~	转换光标当前所在位置字符的大小写。如果一直按住“~”字符键，将会依次转换后续字母的大小写，直至释放“~”字符键
u	撤销最近一次执行的编辑命令，或依次撤销先前执行的编辑命令
u	同上（ex编辑命令）
U	撤销对当前文本行的编辑处理
删除文本	
[n]x	删除光标当前所在位置的字符，或删除从光标当前位置开始的n个字符
[n]X	删除光标当前所在位置的前一个字符，或删除光标当前位置之前的n个字符
dw	删除光标当前所在位置的一个整字或部分字符。如果光标在字首，则删除整字。如果光标在字的中间任何位置，则删除光标位置及之后的字符
[n]dd	删除光标当前所在的文本行，或删除从当前行开始的n个文本行

(续表)

命令	简单描述
D	删除当前文本行从光标位置开始之后的所有字符
dG	删除从当前行开始直至文件最后一行的所有文本行
d[n]G	删除从文件的第n行开始直至当前行的所有文本行
:line#1,line#2 d	删除从指定行号line#1到line#2之间的所有文本行
复制与移动文本	
[n]yy	复制光标当前所在的文本行, 或从当前行开始的n个文本行
[n]Y	同上
p (小写)	把复制或删除 (“dd” 命令) 的文本行粘贴到光标所在文本行的下面
P (大写)	把复制或删除 (“dd” 命令) 的文本行粘贴到光标所在文本行的上面
:line#1,line#2 co line#3	把第line#1 ~line#2行复制到第line#3行之后
:line#1,line#2 m line#3	把第line#1 ~line#2行移至第line#3行之后
设置行号显示	
:set nu	在编辑期间增加临时行号
:set nonu	撤销行号显示 (默认情况)
Ctrl-G	显示当前文件的名称和当前文本行的行号
设置大小写字母检索准则	
:set ic	检索字符串时忽略字母的大小写
:set noic	检索字符串时严格区分字母的大小写 (默认情况)
定位文本行	
[n]G	将光标移至文件的最后一行或第n行
检索与替换	
:/string	向前 (文件结尾方向) 检索指定的字符串
:?string	向后 (文件开头方向) 检索指定的字符串
n	按检索方向找出下一个匹配的字符串
N	逆检索方向找出前一个匹配的字符串
:[g]/search/s//replace/[g][c]	检索并替换字符串
清除屏幕	
Ctrl-L	清除因其他进程的输出信息而干扰的编辑窗口
合并文件与合并行	
:r filename	在光标所在文本行之后插入指定文件的内容
: [line#] r filename	在第line#1行之后插入指定文件的内容
J	把相邻的两个文本行合并为一行 (把下一行合并到光标当前所在文本行的后面)
保存编辑结果与退出vim编辑器	
:w	保存编辑处理后的最终结果 (把内存缓冲区中的数据写到文件中)
:w!	强制保存编辑处理后的结果
:wq	保存编辑处理后的结果, 然后退出vim编辑器

（续表）

命令	简单说明
:wq!	强制保存编辑处理后的结果，然后退出vim编辑器
ZZ	保存编辑处理后的结果，然后退出vim编辑器
:q	在未做任何编辑处理时，可以使用此命令退出vim编辑器
:q!	强制退出vim编辑器，放弃编辑处理后的结果
:w <i>filename</i>	把编辑处理后的结果写到指定的文件中保存
:w! <i>filename</i>	把编辑处理后的结果强制写到指定的文件中保存，即使文件已经存在
:wq! <i>filename</i>	把编辑处理后的结果强制写到指定的文件中保存，即使文件已经存在，然后退出vim编辑器
其他	
:f或Ctrl-G	显示文件的名称、编辑状态、文件的总行数、光标当前所在的行号和列号，以及当前行之前的行数占整个文件总行数的百分比
Ctrl-V	用于输入其他控制字符

第7章 Shell基础知识

本章主要介绍Shell编程的基础知识。我们将从什么是Shell脚本开始，介绍Shell变量与变量替换、命令与命令替换，以及各种test语句。讨论怎样编写和运行Shell脚本，以及怎样利用Shell的编程机制和Linux系统提供的丰富命令，创建自己的Shell脚本。

在本章及随后的第8章中，我们将以Bash Shell为基础，结合Linux系统的日常管理和例行维护，介绍Shell编程的基本概念和技巧。Shell编程涉及的内容非常广泛，而且，如果想把Shell编程的功能发挥得淋漓尽致，还需要对Linux系统有深入的了解。随着对Shell编程和Linux系统的逐渐熟悉，在日常的系统管理和维护过程中，有关的内容自然而然也就掌握了，没有必要从一开始就求大求全。因此，我们采取的原则是介绍Shell编程的原理，重在实用。

7.1 Shell与Shell脚本

在Linux系统的层次组织结构中，Shell是一个重要的组成部分。Shell是用户与Linux系统内核进行联系的桥梁。Linux通过Shell界面，接受用户的请求，利用系统的资源，为用户提供服务。根据Shell的调用方式，Linux系统中的Shell主要分为交互式注册Shell、交互式非注册Shell，以及非交互式Shell三种。

如果调用方式不同，Shell的初始化过程也不相同。在GNOME注册界面或“login:”提示下输入用户名与密码，并成功地注册之后，Linux系统将会调用交互式注册Shell。交互式注册Shell利用/etc/profile和/etc/bash.bashrc文件，以及用户主目录中的~/profile等初始化文件（或称启动文件），设置用户的运行环境。

在Shell命令提示符下输入sh或bash命令，将会进入交互式非注册Shell（也可以看做当前Shell的子Shell）。此时，Shell将会读取并执行/etc/bash.bashrc和~/bashrc等初始化文件，同时还会继承注册Shell利用初始化文件设置的各种环境变量。

非交互式Shell主要用于运行Shell脚本。当利用命令行界面提交一个Shell脚本时，从开始执行到结束，其整个运行环境即为非交互式Shell。非交互式Shell通常不会执行任何用户初始化文件，但会继承注册Shell设置的运行环境，从实际效果来讲，也间接使用或继承了用户初始化文件的运行结果。注意，在开始运行之初，非交互式Shell将会检查BASH_ENV变量。如果变量已经设置，非交互式Shell将会以BASH_ENV变量的值作为初始化文件予以执行。

7.1.1 为什么需要Shell编程

Linux系统提供了丰富的系统命令和各种各样的实用程序，只要经过适当的组合，基本上都可以满足绝大部分应用需求，而不必重新编写新的程序，这是Shell脚本最大的功用所在。实际上，Linux系统的很多系统配置、启动和管理任务都是通过Shell脚本实现的。

Linux操作系统中的Shell既是一个命令解释程序，也是一种强有力的编程语言。作为操作系统内核与用户之间的一个交互界面，Shell可以解释执行用户输入的Linux命令，可以实行I/O重定向，提供管道、元字符匹配以及文件名生成等功能。

作为一种编程语言，Shell也可以执行简单的批处理，利用Linux系统的丰富命令、实用程序和各種工具，完成特定的功能需求。如果这样仍然不能满足需要，还可以利用Shell的编程机制，如测试语句、条件转移和循环控制结构，执行复杂的操作，增强Shell脚本的功能和灵活性。

实际上，借助于Linux系统本身的强大功能，利用Shell的编程机制，很容易实现各种系统管理和日常维护任务，而不需要采用其他高级编程语言，为每一项任务重新开发额外的工具。

在第7章和第8章中，我们将利用大量的例子，介绍Shell的各种功能特性。其中的大部分例子都是从实践中挑选出来的，且经过了验证（注意，使用前应首先利用`chmod`命令，增加Shell脚本的执行权限，然后再运行，参见本章后面的说明）。

从1979年Steve Bourne开发出第一个Shell，且与UNIX系统第7版一同推出开始，Bourne Shell即成为UNIX系统必备的用户界面。随着UNIX系统的不断发展，David Korn在Bourne Shell和C Shell的基础上，开发出Korn Shell之后，进一步增强了Shell的功能。Linux系统中提供的Bash与Bourne Shell和Korn Shell一脉相承，在功能上又有更进一步的提高。

如果想要成为专业水平的Linux系统管理员，熟练地掌握Shell编程技巧是一项必不可少的基本要求，即使不一定非要切实编写Shell脚本。例如，在系统的启动过程中，Linux执行的就是位于`/etc/init.d`目录中的一系列Shell脚本，用以完成系统的各种配置、设置，以及启动各种系统服务等。理解和灵活地运用这些Shell脚本，对于分析系统的行为是非常重要的一项技能。

Shell编程是一种简单直观，能够快速实现复杂功能需求的解决方法。Shell的语法规则简单、直观、易懂，学习和编写Shell脚本并不困难。实际上，只要熟悉Linux命令，了解Shell编程的基本规则和结构语句，利用Shell的编程机制，把若干Linux命令、实用程序或应用程序组合在一起，即可写出自己的Shell脚本。

Shell编程吸取了UNIX/Linux系统把复杂任务分解成简单子任务的优良传统，使应用开发人员能够把各种系统工具和实用程序灵活地组合在一起，完成特定的功能需求。这是一种非常好的解决问题的手段，避免了针对各种各样的用户需求，都要重新开发一套复杂工具的烦恼。事实上，Linux系统已经提供了丰富的、功能强大的系统工具和实用程序，只要灵活地运用这些现成的工具，就可以满足我们的绝大部分需求。

7.1.2 何为Shell脚本

利用文本编辑器，事先把一系列Linux命令或可执行程序放到文件中，然后修改文件的访问权限，使之能够像系统命令或实用程序一样执行，这样的文本文件就是Shell脚本，或称Shell程序。简单地讲，Shell脚本就是一种包含若干Linux命令或可执行程序的文本文件。

Shell脚本可由任何Linux命令组成。当执行Shell脚本时，文件中的所有命令将从头到尾，一个一个地顺序执行（除非Shell脚本中含有控制结构语句）。就像用户在终端前面，以命令行方式，每次输入一个命令，让系统依次执行一样。

同其他面向过程的编程语言一样，除了普通Linux命令或可执行程序之外，Shell脚本还可以包含控制结构和变量，也可以带有参数，使之完成一定的功能需求。

最简单的Shell脚本可以是仅仅包含一个或一组Linux系统命令的文件。利用Shell脚本不仅能够很容易地实现批量处理，避免一遍又一遍地重复输入一系列常用的命令，而且还可以对Shell脚本进行修改或定制，从而满足不同的应用需求或功能需求。

如果再使用变量、控制结构和参数传递等一整套编程机制，灵活地运用Linux系统本身提

供的丰富命令和工具, 在进行Shell编程时无疑会如虎添翼, 从而能够提供更加强大的功能。

7.1.3 运行Shell脚本

Shell脚本有两种执行方式。第一种方式是利用sh或bash等命令(注意, 在Ubuntu 8.10版的Linux系统中不能使用sh, 除非脚本文件的第一行中指定了究竟使用哪一个Shell, 因为此时的sh是dash的符号链接文件, 而dash与bash是不兼容的), 把Shell脚本文件名作为Shell的参数, 由Shell直接读取并解释执行Shell脚本文件中的每一个命令, 这些命令就像直接从键盘上输入一样。这种Shell脚本执行方式只要求Shell脚本文件具有“可读”的访问权限。

假定Shell脚本文件whogrep.sh包含下列一行命令:

```
who | grep cathy
```

采用下列方式运行Shell脚本:

```
$ sh whogrep.sh
```

的效果则相当于执行“who |grep cathy”命令。

第二种执行方式是首先利用chmod命令设置Shell脚本文件, 使Shell脚本具有“可执行”的访问权限。然后在命令提示符下直接输入Shell脚本文件名, 作为一个普通的命令, 交由Linux系统执行。例如:

```
$ chmod 755 whogrep.sh
$ whogrep
cathy    tty7          2009-12-08 09:02 (:0)
cathy    pts/0          2009-12-08 10:09 (:0.0)
$
```

因此, 一旦写好一个Shell脚本(假定为scriptname), 可以采取上述两种调用方式之一。但是, 一般不建议使用第一种调用方式——“sh scriptname”, 尤其不建议采用“sh <scriptname”调用方式, 因为这将禁止Shell脚本读取标准输入。注意, 以“sh scriptname”方式调用一个Shell脚本可能会禁止Shell的部分扩充功能, 因而引起Shell脚本无法正确地执行。

在采用第二种调用方式直接运行可执行的Shell脚本之前, 首先应当使用下列chmod命令之一, 把Shell脚本文件设置为可执行文件:

- chmod 755 scriptname (文件属主可以读、写和执行, 其他用户只能读与执行);
- chmod a+rx scriptname (增加任何用户读与执行的权限);
- chmod u+rx scriptname (仅增加文件属主读与执行的权限)。

按照上述要求设置Shell脚本文件的访问权限之后, 可以采用下列方式, 直接运行Shell脚本:

- scriptname (如果命令检索路径包含当前目录);
- path/scriptname或./scriptname (如果命令检索路径不包含当前目录)。

在测试和调试最终完成之后, 如果愿意, 也可以作为一个工具, 把Shell脚本文件移至某个公用的用户命令目录, 如/usr/local/bin目录中, 使所有的用户均可运行新增的Shell脚本。

7.1.4 退出与出口状态

一个命令或进程终止运行时, 将会自动地向父进程或Shell返回一个出口状态。如果命令或进程成功执行完毕, 将会返回一个数值为0的出口状态。如果命令或进程在执行过程中出现异

常而未正常结束，将会返回一个非零值的出错代码。

同样，一个Shell脚本结束运行时也会返回一个出口状态。在Shell脚本中，最后执行的一条命令将决定整个Shell脚本的出口状态。此外，Shell的内部命令exit也可用于随时终止Shell脚本的执行，并返回Shell脚本的出口状态。

因此，在Shell脚本中，开发人员可以利用“exit [n]”命令，在终止执行Shell脚本的同时，向调用Shell脚本的父进程（也即Shell）返回一个数值为n的出口状态。其中，n必须是一个位于0 ~ 255范围内的整数值。按照惯例，一个命令或Shell脚本正常终止时应返回0，运行有误时可返回一个1 ~ 255范围内的整数值。

当Shell脚本结束运行前执行的最后一个命令是不带任何数值参数的exit语句时，Shell脚本的最终出口状态是Shell脚本执行exit语句之前执行的最后一条命令的出口状态。依据返回的出口状态，Shell可以判断整个Shell脚本的运行是否正常结束。

例如，假定存在下述Shell脚本，command_last命令的返回值就是整个Shell脚本的出口状态：

```
#!/bin/bash
command_1
command_2
.....
command_last
exit
```

在此情况下，其效果等价于最后执行了一个“exit \$?”语句：

```
#!/bin/bash
command_1
command_2
.....
command_last
exit $?
```

在此情况下，也可以干脆去掉exit语句，直接使用最后一条命令的出口状态作为整个Shell脚本的返回值：

```
#!/bin/sh
command_1
command_2
.....
command_last
```

对于顺序执行的Shell脚本，这种做法没有任何问题。但对于带有控制转移和结构语句的Shell脚本而言，则需要在Shell脚本可能终止的任何位置，尽量使用能够返回不同出口状态的“exit [n]”语句，以便了解Shell脚本的实际运行结果。

在任何时候或任何位置，都可以使用Shell的内部变量\$?给出之前执行的最后一条命令的出口状态。Shell函数也是如此，当从函数返回时，\$?变量的值表示函数中实际执行的最后一条命令的出口状态，这也是Shell给出函数返回值的实现方法。

Shell脚本运行终止之后，也可以在Shell命令提示符下，使用\$?内部变量返回Shell脚本的出口状态，也即脚本中执行的最后一条实际命令的出口状态。在Linux系统中，为了测试一个命

令或Shell脚本的执行结果，\$?变量是非常有用的。

下面的例子说明了如何获取命令以及Shell脚本的出口状态：

```
$ cat testexit
#!/bin/bash
LANG=C          # 在英文语言环境中，其说明信息有时更准确一些（本书部分例子中的输出数据
echo hello      # 也是事先设置了英文语言环境后的结果）
echo $?         # 输出脚本中间单个命令的出口状态。这里的返回值为0，因为echo命令总是
                # 能够成功地执行
lsdf            # 不存在的命令
echo $?         # 输出脚本中间单个命令的出口状态。这里的返回值为非0，因为lsdf并不是
                # 一个合法的命令，因而无法执行

echo
exit 100        # 脚本执行到此结束，同时以100作为整个Shell脚本的返回值
$ testexit
hello
0
testexit: line 6: lsdf: command not found
127
$ echo $?       # 脚本终止时再执行此命令，将会返回整个Shell脚本的出口状态（100）
100
$
```

“!”是逻辑非运算符，表示命令返回值或测试结果的否定值，因而也能够影响其出口状态。下面的例子就说明了这个问题：

```
$ true          # true是一个内部命令
$ echo "exit status of \"true\" = $?" # 返回值为0
exit status of "true" = 0
$ ! true
$ echo "exit status of \"! true\" = $?" # 返回值为1
exit status of "! true" = 1
$
```



“!”与true之间必须留有一个空格，否则将会导致系统发出“命令不存在”的出错信息。

7.1.5 调用指定的Shell解释程序

通常，Shell脚本的第一行均包含一个以“#!”为起始标志的文本行，这个特殊的起始标志告诉系统：当前文件包含一组命令，需要提交给指定的Shell解释执行。“#!”特殊标志实际上是一个两字节的文件类型“标识码（magic code）”，表示当前的文件是一个可执行的Shell脚本文件。紧随“#!”标志的是一个路径文件名，指向用于执行当前Shell脚本文件的命令解释程序。

在发现“#!”之后，系统将会采用指定的命令解释程序替代当前的Shell，从第二行开始解释执行脚本中的每一个有效的命令（注释行除外）。指定的命令解释程序可以是一个Shell、一个普通的Linux命令，甚至也可以是一个应用程序。

下面列出的特殊标志行都是合法的，分别表示调用不同的命令解释程序或普通程序，解释执行当前的Shell脚本。其中，“#!/bin/bash”表示调用Linux系统默认的Shell，即Bash。Bash是

Linux系统环境最通用的Shell。“#!/bin/ksh（或#!/usr/bin/ksh）”表示调用Korn Shell解释执行当前的Shell脚本，如此，等等：

```
#!/bin/sh
#!/bin/ksh
#!/bin/bash
#!/bin/tcsh
#!/bin/zsh
#!/bin/more
.....
```

依据Shell脚本中的特殊标志行，当指定的Shell开始解释执行脚本时，将会把标志行作为注释处理。注意，特殊标志后面的路径文件名必须是正确的。否则，系统将会给出命令解释程序有误的错误信息，并立即终止执行当前的Shell脚本。

按照上述说明，如果Shell脚本中存在特殊标志行，则调用适当的Shell解释执行当前的Shell脚本。如果Shell脚本中包含多个特殊标志行，只有第一个标志行起作用。如果Shell脚本中只用到一些常用的系统命令，而没用使用特殊的Shell内部命令，特殊标志行可以省略。

下面的rm file是一个比较特殊的Shell脚本，其中的标志行表示需要调用/bin/rm命令，执行当前的脚本文件。注意，执行这个Shell脚本时不会产生任何输出，尽管其中存在若干echo语句，也不会执行到exit语句，唯一的运行结果（或者说唯一的作用）就是删除这个脚本文件本身。假定当前目录包含两个文件，其中之一就是rm file脚本文件，运行后的结果如下：

```
$ ls -l
总计 8
-rwxr--r--  1 gqxing gqxing 856 2009-11-07 21:10 file2
-rwxr-xr-x  1 gqxing gqxing 111 2009-11-07 21:16 rmfile
$ cat rmfile
#!/bin/rm
echo "This line will never print on screen."
echo "And the current file will be deleted also."
exit 1
$ rmfile
$ ls -l
总计 4
-rwxr--r--  1 gqxing gqxing 856 2009-11-07 21:10 file2
$
```

依照上述脚本，我们可以尝试用“#!/bin/more”作为特殊标志行，创建一个Shell脚本文件，使之显示紧随其后的文本信息。把文件的访问权限修改为可执行文件，然后再执行这个Shell脚本时，将会显示脚本文件中除了标志行之外的所有内容：

```
$ cat readme.sh
#!/bin/more
This is a script to display this file self.
Contents of this file will be displayed
when you excute this script file.
$ readme.sh
#!/bin/more
This is a script to display this file self.
```

```

Contents of this file will be displayed
when you excute this script file.
$

```

7.1.6 位置参数

从命令行上传递给Shell脚本的参数、传递给函数的参数，或通过set命令得到的参数通常称做位置参数。位置参数可以依其出现的位置按顺序号引用（\$0、\$1、\$2、……），故称做位置参数。按照惯例，\$0是Shell脚本文件的名字，由调用Shell脚本的进程（即Shell）负责设置\$0参数。\$1是第一个实际参数，\$2是第二个实际参数，……如此类推。注意，从第10个位置参数开始，必须加用花括号，如\${10}、\${11}和\${12}等。特殊变量\$*和\$@表示所有的位置参数，\$#表示位置参数的总数。

因此，利用特殊的内部变量\$#、花括号和感叹号“!”，可以容易地引用传递给Shell脚本的最后一个位置参数：

```

args=$#           # 传递的参数数量
lastarg=${!args}  # 注意，不能使用“lastarg=${!$#}”直接引用

```

为了改变位置参数的内容，可以使用shift命令。顾名思义，shift命令的功能就是重新分配传递给Shell脚本或函数的位置参数：把参数的位置从右到左依次左移一位，整个位置参数的总数也随之减1。例如，假定位置参数的总数为N，执行shift命令之后，将会发生\$2→\$1、\$3→\$2、……、\$N→\$N-1等变化。原来的\$1和\$N不复存在，但\$0（即Shell脚本的文件名）保持不变。

利用shift命令，可以依次处理每一个位置参数。因此，如果采用适当的循环结构与test语句，即可处理任意数量的位置参数。尽管使用花括号也可以做到这一点，但shift命令更灵活、更方便。

下面是一个利用shift命令和until循环显示所有位置参数的例子。当使用命令行参数“The arguments submitted to the script”调用这个Shell脚本时，通过shift语句，until语句只需检查第一个位置参数是否为空即可遍历每一个位置参数。在依次处理（显示）每一个位置参数之后，整个脚本运行结束：

```

$ cat echoarglist
#!/bin/bash
until [ -z "$1" ] # 表示直至所有的参数均已得到处理再结束
do
    echo -e "$1 \c"
    shift
done
echo
exit 0
$ echoarglist The arguments submitted to the script
The arguments submitted to the script
$

```

每执行一次shift命令，\$@和\$*也会依次舍弃第一个位置参数，保留其余的位置参数。同时，\$#的数量也相应地减1。示例如下：

```

$ cat echoarglist2
#!/bin/bash

```

```

echo "$# ----- @$"
shift
echo "$# ----- @$"
shift
echo "$# ----- @$"
exit 0
$ echoarglist2 1 2 3 4 5
5 ----- 1 2 3 4 5
4 ----- 2 3 4 5
3 ----- 3 4 5
$

```

下面是另一个利用\$*和\$@列出所有位置参数的例子。当使用命令行参数“one two three”调用echoarglist3脚本时，由于\$*与\$@之间的细微差别（参见7.2.3节），且\$*的引用方式不同，在终端窗口上显示的位置参数结果也不完全相同：

```

$ cat echoarglist3
#!/bin/bash
if [ -z "$1" ]
then
    echo "Usage: `basename $0` argument1 argument2 ....."
    exit 1
fi
index=1
echo "Listing arguments with \"\$*\":"
for arg in "$*"          # $*加双引号意味着把所有的参数作为一个单词处理
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Entire argument list is seen as single word."
echo
index=1
echo "Listing arguments with \"\$@":"
for arg in "$@"          # $@加双引号表示把参数看做多个单词
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Argument list is seen as separate words."
echo
index=1
echo "Listing arguments with \$* (unquoted): "
for arg in $*            # 如果$*未加引号，表示把参数看做多个单词
do
    echo "Argument #$index = $arg"
    let "index+=1"
done
echo "Argument list is seen as separate words."
exit 0
$ echoarglist3 one two three
Listing arguments with "$*":
Argument #1 = one two three

```

```
Entire argument list is seen as single word.
```

```
Listing arguments with "$@":
```

```
Argument #1 = one
```

```
Argument #2 = two
```

```
Argument #3 = three
```

```
Argument list is seen as separate words.
```

```
Listing arguments with $* (unquoted):
```

```
Argument #1 = one
```

```
Argument #2 = two
```

```
Argument #3 = three
```

```
Argument list is seen as separate words.
```

```
$
```

在结束这一节的介绍之前, 最后再说明一点。在编写Shell脚本时, 应注意使用模块化的方法组织Shell脚本。在学习Shell编程的过程中, 还应随时注意多积累, 多收集一些“模型”代码片段, 这对将来编写Shell脚本是非常有用的, 甚至可以考虑建立一个代码片段储备库。例如, 下列代码片段就是一个比较好的通用开场白, 其目的是测试调用Shell脚本时是否提供了正确数量的参数:

```
if [ $# -ne $Number_of_expected_args ]
then
    echo "Usage: `basename $0` script_arguments"
    exit 1
fi
```

7.2 变量与变量替换

变量是每一种编程语言和脚本语言中都不可缺少的重要组成部分之一。在Shell中, 变量可用于字符串操作、算术运算、参数传递以及其他运算。实际上, 变量无非是系统分配的一个或一组内存位置, 用以存储各种数据。

除了特殊字符, Shell变量名可以由任何字母、数字和下画线等字符组成, 但第一个字符必须是字母或下画线。变量名的长度没有限制。

与其他编程语言不同的是, Shell并不区分变量的类型。实际上, Shell仅支持一种类型的变量, 即字符串变量。也就是说, Shell中的所有变量都是字符串类型的。但Shell对字符串变量的处理并不仅限于字符串操作。

在Shell的处理过程中, 取决于上下文以及采用的运算符和命令工具, Shell允许对变量执行诸如整数算术运算等多种操作。能够影响变量类型的另外一个因素是变量值的首字符。如果首字符为数字, 表示还可以对变量执行整数算术运算, 否则只能执行字符串操作。

7.2.1 变量分类

从用途上考虑, 变量可以划分为内部变量、本地变量、环境变量、参数变量和用户定义的变量。内部变量是为便于Shell编程而由Shell设定的变量, 如表示错误类型的ERRNO变量等, 详见7.2.3“内部变量”一节。在代码块或函数中定义的变量, 且仅在定义的范围内有效的变量称做本地变量。环境变量是为系统内核、系统命令和应用程序提供运行环境而设定的变量, 常

见的有PATH、HOME和LANG等变量。参数变量指调用Shell脚本或函数时传递的变量（因此，在Shell或Shell脚本中，变量替换也称参数替换）。用户定义的变量是为运行用户程序或为完成某种特定的任务而设定的普通变量或临时变量。

广义地讲，每个进程都有一个运行“环境”，这个运行环境将影响进程的执行行为。Shell也存在一组可以引用的环境变量，其中的每个变量都被赋予一定的信息。从这个意义上讲，Shell同其他任何进程是一样的。

每当开始运行时，Shell都会创建一组自己的环境变量。更新或增加新的环境变量将会引起Shell更新自己的运行环境。如果在Shell脚本中设置了环境变量，通常需要使用export命令予以公布，以便子进程能够继承。

**注意**

环境变量的空间分配是有一定限制的。建立太多的环境变量将会占用额外的空间，因而可能会引起问题。

7.2.2 变量赋值

变量的赋值可以采用赋值运算符“=”实现，其语法格式如下：

```
variable=value
```

在Shell中，等号“=”是一种通用的赋值运算符，可用于数字和字符串赋值。变量赋值也可用于声明变量，对变量进行初始化，或改变变量的值。另外，为使其他Shell脚本或程序能够使用定义的变量，需要利用“export variable”命令，把变量导至Shell运行环境。注意，赋值运算符“=”前后不能有空格。例如，为了定义一个EDITOR环境变量，可以使用下列命令：

```
$ EDITOR=vi; export EDITOR
$
```

在变量的赋值过程中，如果赋的值是由多个单词组成的字符串，应在字符串前后增加一对单（或双）引号，以便Shell能够把字符串作为一个值处理。例如：

```
$ TITLE="Name Phone Email-Address"; export TITLE
$
```

未初始化的变量（即未赋值的变量）的值为“null”（注意，使用未赋值的变量有时会出现问题）。利用下列变量赋值形式，即可声明一个未初始化的变量：

```
variable=
```

另外一种赋值方法是利用read语句在执行过程中为变量赋值，参见第7.3.2节有关read语句的介绍；此外，还可以在for循环结构语句中，采用“for variable in word-list”的形式为变量赋值，详见第8章“Shell高级编程”。

在Shell脚本中，适当地使用变量能够提高脚本的可读性和灵活性，确保数据的一致性，而且便于维护。

7.2.3 内部变量

Shell提供一组丰富的内部变量，能够为用户的Shell编程提供支持。熟练地掌握Shell内部变

量，对以后的Shell编程有极大的帮助。为了能够对Shell内部变量的概貌有一个全面的了解，也为了阅读本书的方便，以及在实际应用中作为参考，表7-1给出了部分常用的主要内部变量。

表7-1 Shell自动设置的内部变量

变量	简单说明
#	\$#变量是命令行参数或位置参数的数量
-	\$-变量是传递给Shell脚本的执行标志（参见set内部命令）
?	\$?变量是最近一次执行的命令或Shell脚本的出口状态
\$	\$\$变量是Shell脚本的进程ID。Shell脚本经常使用\$\$变量组织临时文件名，确保文件名的唯一性
_	\$_是一个特殊的变量，在Shell开始运行时，变量的初始值为Shell或其执行的Shell脚本的绝对路径名，之后就是最近执行的命令的最后一个选项或参数等。例如，注册后打开一个终端窗口，接着执行一个du命令时，\$_变量的值如下： \$ echo \$_ /etc/bash_completion \$ du /usr/bin /usr/sbin 161756 /usr/bin 41056 /usr/sbin \$ echo \$_ /usr/sbin \$
!	#!变量的值是最近运行的一个后台进程的PID
*	\$*变量表示所有的位置参数，其值是所有位置参数的值，每个参数均可看做一个单独的字。引用时，\$*相当于\$1、\$2、\$3、…，表示多个参数。而“\$*”则相当于“\$1 \$2 \$3 …”，表示一个参数
@	\$@变量类似于\$*，表示所有的位置参数，其值也是所有位置参数的值，每个参数均可看做一个单独的字。引用时，\$@相当于\$1、\$2、\$3、…，表示多个参数。但“\$@”则相当于“\$1”、“\$2”、“\$3”、…，每个参数均为前后加引号的字符串。也就是说，参数是原封不动进行传递的，未做解释或扩展
LINENC	Shell脚本中当前执行的命令的行号。仅当调试Shell脚本时，这个变量才有意义
OLDPWD	利用cd命令改换到新目录之前所在的工作目录
OPTARG	getopts命令已经处理的前一个选项参数
OPTIND	getopts命令已经处理的前一个选项参数的索引
PPID	\$PPID是当前进程的父进程的PID
PWD	表示当前工作目录，其变量值等同于pwd命令的输出
RANDOM	每次引用这个变量时，即可生成一个均匀分布于0~32 767范围内的随机整数。如果赋予RANDOM变量一个数值，可以达到初始化随机数序列的目的
REPLY	当使用read命令读取输入数据时，如果没有指定变量参数，可以把REPLY变量用做read命令的默认变量，从而把read命令读取的输入数据赋予REPLY变量
SECONDS	\$SECONDS变量是脚本已经运行的时间（秒数）

表7-2给出了Shell使用的部分主要环境变量。

表7-2 Shell使用的环境变量

变量	简单说明
BASH_ENV	当调用非交互式的Bash运行Shell脚本时，如果这个变量已经设置，Shell将会使用这个变量值指定的绝对路径文件名，作为可在当前环境中执行的初始化文件。典型情况下， <code>profile</code> 等初始化文件是在会话启动阶段执行的，而BASH_ENV变量指定的文件是在Shell运行的初始阶段执行的。对于BASH_ENV变量指定的文件，Shell采取与点“.”命令类似的方法解释执行，即在当前环境中执行其中的命令。另外，BASH_ENV变量指定的文件只要具有可读的访问权限即可，不必是可执行文件。但与使用点“.”命令执行脚本不同的是，执行BASH_ENV变量指定的文件时不会使用PATH变量检索命令文件，这主要是出于安全考虑。在设置BASH_ENV变量值时，其中可以包含变量替换和命令替换等
CDPATH	定义cd命令的检索路径。如果cd命令中指定的目的目录为相对路径名，cd命令首先会在当前目录“.”中搜寻目的目录。如果未发现目的目录，则开始检索CDPATH变量中列举的每一路径名，直至找到目的目录，并成功地改换工作目录。如果最终仍未发现目的目录，则保持当前工作目录不变。例如，假定把CDPATH变量设置为/home/gqxing，其中存在两个子目录bin和src。如果用户当前位于/home/gqxing/bin目录中，输入“cd src”命令后，即使没有指定全路径名，仍可把工作目录改换到/home/gqxing/src中
COLUMNS	用于定义终端窗口的列宽。select内置命令使用此变量值确定数据显示的宽度，以便输出菜单选项列表。此变量值也用于确定Shell编辑窗口的列数
EDITOR	用于确定命令行编辑使用的编辑程序，通常可以设为vi或emacs等用户熟悉的编辑器
FCEDIT	用于设定fc内置命令使用的默认编辑器，以便用户能够使用熟悉的编辑器编辑命令行
HISTFILE	用于指定存储命令历史记录的文件，默认的文件为~/bash_history
HISTFILESIZE	用于设定命令历史文件保存的最大命令记录数量，默认值为500（条命令）
HISTSIZE	用于设定命令历史缓冲区保存的最大命令记录数量，默认值为500（条命令）
HOME	用户主目录的路径名（也是cd命令的默认参数）。普通用户的主目录通常为/home/username，超级用户的主目录为/root
IFS	字段分隔符（默认值为空格、制表符和换行符）。字段分隔符通常用于解析命令行或字符串的基本构成元素。简言之，字段分隔符决定了Shell在解析命令行或字符串时怎样确定字段或单字等基本元素的边界。IFS变量值中的第一个字符用于解析\$*变量中的位置参数
INPUTRC	用于设定readline启动文件的名字，默认值为~/inputrc
LANG	用于设置语言环境，参见第9章“用户管理”
LC_ALL	用于统一设置LC_*系列变量的值
LC_CTYPE	用于确定系统怎样处理各种语言环境的字符集，包括字符的分类，字符的大小写转换规则，以及其他字符属性
LC_MESSAGES	用于确定采用何种语言输出系统提示信息
LC_NUMERIC	用于确定本地化千分数值的显示格式
LINES	用于定义终端窗口的行数。select内置命令使用此变量值确定数据显示的高度，以便输出菜单选项列表。变量值可用于确定Shell编辑窗口的行数
MAIL	用于定义用户邮箱的路径文件名
MAILCHECK	指定Shell检测邮件的频度（以秒为单位），默认值为60秒。如果未设置这个变量，或把变量设置为小于或等于0的数值，Shell将停止检测邮件
MAILPATH	定义系统检查是否有新邮件到来的文件名

(续表)

变量	说明
PATH	指定命令的检索路径及顺序（普通用户的命令检索路径通常包括/bin和/usr/bin等目录，超级用户的命令检索路径通常包括/sbin、/bin、/usr/sbin和/usr/bin等目录）。当用户输入命令时，Shell将会根据PATH变量列举的目录及顺序，从中检索并执行匹配的可执行程序。如果提交的命令其目录不在检索路径中，必须输入命令的完整路径名才能执行。另外，命令检索路径的顺序是非常重要的。当检索路径中的不同目录存在同名的命令时，Shell将会使用第一个发现的命令。例如，假定PATH变量定义为PATH=/bin:/usr/bin:/usr/sbin:\$HOME/bin，且存在两个sample命令文件，分别位于/usr/bin和/home/gqxing/bin目录时，如果输入sample命令时未指定完整的路径名，Shell将会调用/usr/bin目录下的sample命令。定义PATH环境变量时，目录之间需加冒号“:”分隔符。通常，PATH环境变量是由/etc/profile及\$HOME/profile等初始化文件设定的。在Shell脚本中，也可以把某个目录临时加到命令检索路径中。一旦终止脚本的运行，立即恢复到之前的PATH变量设置（作为一个子进程，Shell脚本不能改变父进程Shell的运行环境）。注意，从安全的角度考虑，\$PATH通常不应包含当前工作目录
PPID	表示父进程的PID
PS1	第一级Shell命令提示符，或称主提示符。变量的默认值为“[\u@h \W]\$ ”，其中的“\”表示普通用户的命令提示符为“\$”，超级用户的命令提示符为“#”。详见第9章“用户管理”
PS2	第二级Shell命令提示符，默认值为大于号“>”。如果输入的命令不完整，或需要用户提供附加的数据时，系统将会按此变量的设置提示用户继续输入
PS3	第三级命令提示符，其默认值为“#?”。这个变量主要用于设置select循环控制结构使用的菜单选择提示符。详见第8章“Shell高级编程”
PS4	第四级命令提示符，其默认值为加号“+”。这个变量主要用做Shell脚本的调试标志符。在跟踪脚本执行的过程中，Shell将会在显示其执行的每一个命令之前，首先输出这个变量定义的数值。详见第8章“Shell高级编程”
SHELL	Shell命令文件的完整路径名。vi/vim等工具使用这个变量值作为默认的Shell
TMOUT	用于定义用户与系统会话过程的超时值。在一个交互式的Shell中，TMOUT变量值可以解释为自输出命令提示符之后等待用户输入的时间（以秒为单位）。Shell输出命令提示符之后，如果在TMOUT规定的时间之内未输入任何命令，Shell将会因超时而终止执行，导致系统关闭用户的终端窗口，或者断开用户的终端连接。对于read内置命令而言，如果TMOUT变量值大于0，这个数值可以用做read内置命令默认的超时值。对于select内置命令而言，如果在TMOUT规定的时间之内一直没有收到输入数据，select命令将会终止执行

7.2.4 变量的引用与替换

使用变量的主要目的是存储或引用其中的数据值。在大型应用程序、Shell或Shell脚本中，经常使用变量，引用变量中的数值。引用变量中的数值称做变量替换。当变量用做参数时，其中的变量替换也称参数替换。

假定variable是一个变量，在变量名字前面加上一个美元符号“\$”前缀即可引用变量的值，也即使用变量中存储的数值替换变量名字本身。在Shell中，引用变量的值（或者说变量替换）主要有以下三种形式：

- \$variable
- \${variable}

- "\$variable" 或 "\${variable}"

但在下列情况下，引用变量时无需在变量名字前面加美元符号“\$”前缀：

- 声明语句；
- 赋值语句；
- unset、export命令；
- 引用系统定义的信号。

请注意区分变量的名字及其引用的数值。通常，如果variable是变量的名字，那么\$variable、\${variable}、"\$variable"或"\${variable}"就表示引用变量的值。例如，在执行下列变量赋值语句之后，echo命令中的变量引用\$variable意味着显示其对应的数值100：

```
$ variable=100
$ echo $variable
100
$
```

在下列变量设置中，第一个PATH变量定义了命令的检索路径，第二个PATH变量定义采用变量替换的形式，重新定义了PATH变量，把PATH变量的值以及用户主目录下的子目录bin和当前目录赋值到PATH变量中：

```
$ PATH=/bin:/usr/bin:/sbin:/usr/sbin
$ PATH=$PATH:$HOME/bin:.
$
```

执行变量替换后，新的命令检索路径如下：

```
$ echo "Now the PATH is $PATH"
Now the PATH is /bin:/usr/bin:/sbin:/usr/sbin:/home/gqxing/bin:.
$
```

为了避免引起歧义，可以采用第二种变量替换形式引用变量。实际上，第一种变量替换形式只是第二种变量替换形式的简化。例如，我们可以把上述PATH变量的定义改写为：

```
PATH=${PATH}:${HOME}/bin:.
```

第三种变量引用方式是在第一种或第二种变量引用方式的基础上加上双引号。这也是一种最保险的变量引用方式。注意，在某些情况下，尤其是传递参数时，第一种或第二种变量引用形式并不可靠，有时还会产生意想不到的错误。例如，假定存在一个简单的Shell脚本，其功能是显示并处理接收的位置参数：

```
$ cat disparg
#!/bin/bash
echo $1
exit 0
$
```

同时又声明了一个var变量，其中含有一个由5个英文单词组成的字符串：

```
$ var="The variable contains five words"
$
```

如果分别采用上述三种变量引用形式显示同一变量的值，其输出结果如下：

```
$ disparg $var
The
$ disp ${var}
The
$ disp "$var"
The variable contains five words
$
```

双引号括住的参数可以看做一个单词，即使其中包含空格分隔符。因此，如果希望把包含多个单词的字符串看做一个参数，应注意使用双引号，以防止单词的分割。

由此可以得出一个结论：当一个变量的值是由一个不包含任何字段分隔符的整体字符串或数值组成时，上述三种变量引用方式的最终结果是完全一样的。但是，如果变量的值包含空格、制表符、换行符，以及由内部变量IFS定义的字段分隔符时，只有第三种变量引用方式才是最保险的。在以后的Shell脚本编程过程中，这一点应切实记住。

此外还要注意，使用双引号引用变量时可以进行替换，如果使用单引号，则意味着按文字引用其中的字符串，即使存在变量引用也不能进行替换。单引号表示把特殊字符\$和变量名看做文字，禁止引用其中的变量值，因而不会出现变量替换。也就是说，单引号中的每个字符，包括特殊字符均做文字常量解释。

7.2.5 变量的间接引用

假定一个变量的值是另一个变量的名字，利用第一个变量是否能够引用第二个变量的值呢？例如，如果a=b，而b=c，仅仅引用变量a是否能够返回变量b的值（c）呢？回答是肯定的。实际上，采用“eval var1=\\${var2}”命令即可达到上述目的。这种情况称做变量的间接引用。

下面是一个间接引用变量的例子：

```
$ Message=Hello
$ Hello="Good Morning"
$ echo "Message = $Message"           # 直接引用
Message = Hello
$ eval Message=\${Message}           # 间接引用
$ echo "Now message = $Message"
Now message = Good Morning
$
```

如果把上述的最后两个命令合并到一起，其效果如下：

```
$ Message=Hello
$ Hello="Good Morning"
$ echo Message = $Message
Message = Hello
$ eval echo Now message = \${Message}
Now message = Good Morning
$
```

在Bash中，还可以采用下列方式实现变量的间接引用：

```
$ Message=Hello
```

```
$ Hello="Good Morning"
$ echo ${Message}
Hello
$ echo ${!Message}
Good Morning
$ Hello="Hello again"
$ echo ${!Message}
Hello again
$
```

7.2.6 特殊的变量替换形式

在Shell中，为了保证Shell、Shell脚本或应用程序能够正常地运行，Shell提供一种特殊的变量替换机制。特殊变量替换的主要功能如下。

- 对于未设置的变量，采用特殊替换表达式赋予默认值；
- 采用特殊替换表达式替换或设置默认值；
- 采用特殊替换表达式给出错误提示信息。

表7-3 给出了Shell支持的部分常用特殊变量替换形式。

表7-3 特殊的变量替换

特殊变量替换	简单说明
<code>\${var}</code>	其作用同 <code>\$var</code> 一样，表示变量 <code>var</code> 的值。在某些情况下，只有采用 <code>\${var}</code> 变量引用形式，其意义才是比较明确的。例如，在定义 <code>PATH</code> 环境变量时，如果需要连接字符串变量， <code>PATH=\${PATH}:/opt/bin</code> 比 <code>PATH=\$PATH/opt/bin</code> 更易于阅读和理解
<code>\${var:-value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>value</code> 作为变量 <code>var</code> 的值进行变量替换。否则，使用变量 <code>var</code> 的值进行变量替换，但变量 <code>var</code> 的值保持不变
<code>\${var:=value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，把 <code>value</code> 赋予变量 <code>var</code> ，同时执行变量替换
<code>\${var:+value}</code>	如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>null</code> 进行变量替换。如果变量 <code>var</code> 已经设置，则使用 <code>value</code> 进行变量替换，变量 <code>var</code> 的值保持不变
<code>\${var:?value}</code>	如果变量 <code>var</code> 已经设置，使用变量 <code>var</code> 的值进行变量替换。如果变量 <code>var</code> 未设置或其值为 <code>null</code> ，使用 <code>value</code> 作为错误提示信息。如果省略了 <code>value</code> ，则输出默认的错误信息，表示变量 <code>var</code> 未设置。然后，终止Shell脚本的执行，并返回一个非0的出口状态（交互式的Shell会话例外）。变量 <code>var</code> 的值保持不变

在上述表达式中，冒号“:”意味着需要测试给定的变量`var`是否已经设置。如果变量已经设置（声明），再检查其值是否为`null`。如果省略了冒号“:”，意味着只需检查`var`是否尚未设置。表7-4总结了各种变量替换形式中有无冒号“:”的细微差别之处。

表7-4 各种变量替换形式的比较

比较	var已设置且其值为非null	var已设置但值为null	var未设置
<code>\${var:-value}</code>	使用 <code>var</code> 替换	使用 <code>value</code> 替换	用 <code>value</code> 替换
<code>\${var-value}</code>	使用 <code>var</code> 替换	使用 <code>null</code> 替换	使用 <code>value</code> 替换
<code>\${var:=value}</code>	使用 <code>var</code> 替换	使用 <code>value</code> 赋值并替换	使用 <code>value</code> 赋值并替换

(续表)

比较	var已设置且其值为非null	var已设置但值为null	var未设置
<code>\${var=value}</code>	使用var替换	使用var替换	使用value赋值
<code>\${var:?value}</code>	使用var替换	错误, 退出, 返回1	错误, 退出, 返回1
<code>\${var?value}</code>	使用var替换	使用null替换	错误, 退出, 返回1
<code>\${var:+value}</code>	使用value替换	使用null替换	使用null替换
<code>\${var+value}</code>	使用value替换	使用value替换	使用null替换

例如, 假定变量var已经设置, 但其值为null, 则`${var:-undefined}`变量替换后的结果是undefined, 示例如下:

```
$ var=
$ echo ${var}

$ echo var is ${var:-undefined}
var is undefined
$
```

在调用Shell脚本时, 如果提供的命令行参数不足, 采用上述默认的变量或参数替换方法很有用。例如, 如果命令行中未给定文件参数, 下列Shell脚本中的command命令(任何合法的文件操作命令)将会对run.data文件进行处理:

```
$ cat substitute
#!/bin/bash
def_fname=run.data
fname=${1:-$def_fname}
command $fname
exit 0
$
```

在下面的例子中, 变量var的值为null。因此, `${var:=abc}`变量替换后的结果是: 首先把abc赋予变量var, 然后使用变量var的值(abc)进行变量替换

```
$ unset var
$ echo ${var:=abc}
abc
$
```

在下面的例子中, 变量var的值为null, 且“?”后面为空。因此, `${var:=abc}`变量替换的结果是输出一条默认的出错信息: “parameter null or not set”。

```
$ unset var
$ echo ${var:?}
bash: var: parameter null or not set
$
```

set命令可用于设置参数。在下面的例子中, set命令设置的位置参数\$1、\$2和\$3, 其值分别为a、b和c。如果使用`${3:+posix}`进行变量替换, 其结果是posix:

```
$ set a b c
```

```
$ echo $3
c
$ echo ${3:+posix}
posix
$
```

在上述的特殊变量替换形式中，变量名后面的参数不仅可以是任何字符串，还可以是变量替换、命令替换和算术表达式的计算结果。

假定之前并未设置username变量，下列变量替换的结果是“whoami”命令的输出：

```
$ echo ${username-`whoami`}
gqxing
$
```

在下面的例子中，只要变量username已经声明，不管其值设置与否，变量替换的最终结果总是“who am i”命令输出的第一个字段：

```
$ username=
$ echo ${username+`who am i | cut -d' ' -f1`}
gqxing
$
```

在下面的例子中，仅当变量var未声明，或即使已经声明但变量值为null，才使用pwd命令的输出（关于\$(pwd)形式的命令替换，参见7.3节“命令与命令替换”）替换变量var的值：

```
$ echo ${var:-$(pwd)}
/home/gqxing
$
```

表7-5给出了其他有用的特殊变量替换形式。

表7-5 其他特殊变量替换形式	
其他特殊变量替换	简单说明
<code>\${#var}</code>	字符串的长度（即字符串变量var中的字符数量）。对于数组来讲， <code>\${#array}</code> 是数组中第一个元素的长度。注意下列例外情况： <code>\${#*}</code> 和 <code>\${#@}</code> 给出的是位置参数的个数，而针对数组的 <code>\${#array[*]}</code> 和 <code>\${#array[@]}</code> 给出的是数组元素的个数
<code>\${var#Pattern}</code> 、 <code>\${var##Pattern}</code>	从\$var变量值的前部删除与给定模式匹配的最短或最长部分子串。如果应用于文件路径名， <code>\${var##Pattern}</code> 的效果相当于basename命令
<code>\${var%Pattern}</code> 、 <code>\${var%%Pattern}</code>	从\$var变量值的后部删除与给定模式匹配的最短或最长部分子串。其中， <code>\${var%Pattern}</code> 可用于抽取路径名的目录部分

在下面的例子中，变量DOCS已经声明，且设置为/docs/posix，变量替换的结果是给出\$DOCS变量值的字符串长度：

```
$ DOCS=/docs/posix
$ echo ${#DOCS}
11
$
```


在下面的例子中，变量var已设置为file.c，变量替换的结果是从整个文件名中抽取除“.c”后缀之外的文件名部分：

```
$ var=file.c
$ echo ${var%.c}.o
file.o
$
```

在下面的例子中，变量var已经声明并设置为posix/src/std，变量替换的结果是从路径名后部删除首次出现并包含斜线字符“/”的最长字符串：

```
$ var=posix/src/std
$ echo ${var%/*}
posix
$
```

在下面的例子中，变量var已设置为\$HOME/src/cmd，变量替换的结果是抽取整个路径名除\$HOME变量值之外的目录名部分：

```
$ var=$HOME/src/cmd
$ echo ${var#$HOME}
/src/cmd
$ echo ${var#$HOME/}
src/cmd
$
```

在下面的例子中，变量var设置为/one/two/three，变量替换的结果是抽取路径名的文件名部分：

```
$ var=/one/two/three
$ echo ${var##*/}
three
$
```

在下面的例子中，变量var设置为/one/two/three，变量替换的结果是抽取路径名的目录部分：

```
$ var=/one/two/three
$ echo ${var%/*}
/one/two
$
```

7.2.7 变量声明与类型定义

前面说过，Shell并不严格区分变量的类型，一切取决于变量存储的内容。在第一次引用时，Shell将根据变量的内容确定变量的类型。为了加快Shell的运算速度，提高Shell编程的严谨性，在Bash中，可以使用typeset或declare命令定义变量的类型，并在定义时对变量进行初始化。

变量无类型之分既是好事，也是坏事。这虽然增加了脚本编程的灵活性，简化了代码的编写，但也容易产生潜在的错误，容易给自己套上绳索，造成之后阅读的困难，养成不好的编程习惯。

**注意**

记住脚本变量的类型是Shell编程人员的责任，不要指望Shell做类型检查。

typeset内部命令（declare内部命令的使用说明详见第8章）用于设定变量的属性。利用typeset命令的“-i”选项，可以把变量声明为整数变量。在声明变量的同时，也可以为变量赋值，进行初始化。事先声明为整数的变量可以直接执行算术运算，而不需要使用expr和let命令，例如：

```
$ typeset -i number          # 把变量number声明为整数变量
$ number=3                  # 之后可以使用整数为变量number赋值
$ echo "Number = $number"
Number = 3
$ number=three              # 但不能使用字符串为变量number赋值
$ echo "Number = $number"
Number = 0
$
```

如果事先并未把变量声明为整数变量，Shell将会把赋予变量的值看做字符串，例如：

```
$ n=6/3
$ echo "n = $n"
n = 6/3
$ typeset -i n
$ n=6/3
$ echo "n = $n"
n = 2
$
```

利用“-r”选项还可以把变量声明为只读变量。此后，任何试图修改只读变量的操作将会失败，并产生错误信息。如“typeset -r var”将会把变量var声明为只读变量。

7.3 命令与命令替换

7.3.1 Shell内部命令

Linux系统提供大量的命令供用户使用。但出于性能方面的考虑，Shell也提供若干具有同样功能的内部命令。内部命令的执行速度比外部命令要快得多，这是因为在解释执行内部命令时，Shell不需要创建子进程。而执行外部命令时需要创建单独的新进程，从而加大了系统的开销。

某些Shell内部命令与系统命令具有相同的名字，但其实现方式不同。例如，Shell内部命令echo与系统命令/bin/echo的名字完全一样，功能也相同，但其执行效率并不相同。

例如，下面两条命令的作用都是在终端窗口上显示一条信息：

```
$ echo "This line uses the \"echo\" builtin to output."
This line uses the "echo" builtin to output.
$ /bin/echo "This line uses the /bin/echo system command to output."
This line uses the /bin/echo system command to output.
$
```

表7-6 给出了Shell提供的部分主要内部命令及说明。

表7-6 Shell内部命令

内部命令	简单说明
.	用于从命令行中读取Shell脚本，并在当前的Shell环境中执行。在Shell脚本中使用点命令时，可以把指定的源文件读入当前脚本中，并从当前位置开始执行。当多个Shell脚本共用同一个数据文件或函数时，点命令是非常有用的。点命令最重要的用途也许就是执行环境变量设置脚本。在此情况下，即使退出环境变量设置脚本，环境变量的设置仍然保持有效。如果直接运行环境变量设置脚本，在脚本终止运行之后，脚本中设置的环境变量也将随之消失而不复存在
:	null语句。本身不做任何处理动作，但总是返回一个值为0的出口状态。其功能一是用做无限循环语句的测试条件；二是作为一种特殊的命令形式，引起Shell处理紧随其后（与“:”位于同一行）的变量替换或命令替换
alias	alias命令用于设置或显示系统或用户定义的命令别名
bg	把指定的作业置入后台运行模式。如果未给定作业号，则把当前作业置入后台运行模式
break	用于退出for、while和until等循环语句
cd	改变目录。用于转换到指定的目录
continue	用于结束for、while和until等循环语句中的当前循环，并立即开始执行下一轮循环
declare	用于声明变量或定义变量的属性。如果未指定变量名，declare命令将会给出所有的变量定义。 “-p”选项用于显示给定变量的值和属性。下列选项用于声明变量，并限定变量的属性： -a 把指定的变量定义为数组变量 -f 用于声明一个函数或显示函数定义 -i 把指定的变量定义为整数变量。当赋予变量值时，按算术扩展的要求执行算术运算 -r 把指定的变量定义为只读变量。在随后的处理过程，不能为只读变量赋值 -x 公布指定的变量，以便其他命令或Shell脚本能够继续使用
echo	echo命令用于输出字符串、变量值或表达式的计算结果等参数。通常，echo命令会在输出信息之后附加一个换行字符
enable	启用和禁用Shell内置命令。禁用Shell内置命令，使得用户无需指定完整的路径名即可运行与Shell内置命令同名的命令，尽管Shell内置命令优先于PATH变量指定的检索目录中的命令。例如，为了运行一个非Shell内置命令的test命令（/usr/bin/test），可以使用“enable -n test”命令禁用Shell内置命令test
eval	用于读取并组合随后的命令参数，然后提交Shell执行
exec	如果exec命令带有参数，则使用命令参数代替当前的Shell进程，而不是像通常那样创建一个新进程，执行给定的命令。如果在Shell脚本中使用exec命令，当由exec命令引入的命令终止运行时，将会强制Shell脚本也随之终止执行。因此，如果在脚本中使用exec命令，通常应作为最后一条命令。如果未带参数，exec命令的作用只是按照I/C重定向的要求，修改文件描述符。例如，“exec <fname”命令意味着使用给定的文件fname代替标准输入
exit	用于无条件地终止Shell脚本的执行。exit命令可以带一个整数参数，作为Shell脚本的出口状态返回Shell。如果使用单独的exit命令（省略整数参数）终止脚本的执行，整个Shell脚本的出口状态则是在exit之前执行的最后一条命令的出口状态。一个好的编程习惯是在脚本执行结束处增加一条“exit N”命令，表示运行成功或出现的不同错误情况。此外，文件结束标志也会引起Shell脚本终止执行
export	export命令用于公布或导出设置的变量，使变量的设置能够用于正在运行的脚本或Shell的所有子进程。export命令的一个重要用途是在profile文件中初始化环境变量，使随后的所有进程都能够访问

内部命令	简单说明
fc	fc命令用于列出、编辑或重复执行命令历史缓冲区或文件中记录的命令
fg	把指定的作业置入前台运行模式。如果未给定作业号，则把当前作业置入前台运行模式
getopts	用于解析传递给Shell脚本的命令行参数。getopts使用两个变量：其中的OPTIND是指向下一个命令行选项的指针；OPTARG是与选项有关的参数。getopts命令通常主要用于while循环语句，通过循环方式处理所有的选项和参数
hash	用于记录给定命令的路径名，以便Shell或Shell脚本随后调用hash命令时不必再按PATH变量指定的目录检索命令。当调用hash命令而未给定参数时，系统仅会显示当前Shell散列表中记录的命令
jobs	显示指定或全部活动的作业。其中的“-l”选项表示显示附加的进程ID信息，“-n”选项表示仅显示已经停止或终止运行的作业
kill	用于向指定的进程或作业发送信号，从而终止相应的进程或作业。信号可以是一个数字代码或信号名字
let	let命令用于实现算术运算
logout	退出注册Shell
printf	按照给定的控制格式输出参数的值。输出格式是一个字符串，其中包括三种字符对象：一为普通字符，可直接输出到标准输出；二为转义字符序列，用于输出特殊字符；三为格式规范，用于定义参数值的输出形式。参见标准的printf()格式定义
pwd	显示当前的工作目录。其作用等同于读取PWD变量的值。换言之，pwd命令替换的结果等同于PWD变量的值
read	用于从标准输入中读取数据，并把数据赋值到给定的变量中。也就是说，read命令能够以交互方式读取来自键盘的输入数据。“-r”选项意味着忽略转义符号“\”，使之仅作为普通字符处理
readonly	把指定的变量设置为只读变量，因而不允许在随后的操作中对相应的变量进行赋值。如果试图修改相应的变量，将会产生一个错误信息
return	引起Shell函数返回主程序的调用点。在非Shell函数的情况下，return命令相当于exit命令
set	<p>set命令用于改变内部变量或脚本变量的值。set命令的一个重要用途是重置位置参数（例如，set ‘command’）。如果不带任何参数，set命令将会输出所有环境变量的当前设置，这是set命令的另一个重要用途。set命令的部分选项说明如下：</p> <ul style="list-style-type: none"> -a 使定义的所有变量和函数设置能够自动导入Shell运行环境 -e 如果其中任何一条命令结束后返回非零值的出口状态，立即终止Shell脚本的执行 -f 禁用文件名生成机制 -m 监控模式（启用作业控制功能）。在交互式Shell中，这个选项的默认设置为on -n 读取Shell脚本中的命令，检查是否存在语法错误，但不执行 -o 使用下列参数设置各种标志： <ul style="list-style-type: none"> • emacs 使用emacs作为命令行编辑器。在交互式Shell中，自动启用emacs作为默认的命令行编辑器，除非调用Shell时指定了“--noediting”选项 • noclobber 防止标准输出(>)重定向覆盖现有的同名文件。一旦打开此标志，仅当使用“>”时才能清除现有的文件 • nolog 不允许把函数定义写入命令历史文件中 • verbose 同“-v”选项 • vi 使用vi/vim作为命令行编辑器。在vi/vim命令行编辑环境中，一开始总是处于命令模式 • 如果不加任何参数，set命令将会输出当前的各种标志设置

(续表)

内部命令	简单说明
	<ul style="list-style-type: none"> -t 读入并在执行任何一条命令之后立即终止Shell脚本的运行 -u 执行变量替换时, 事先未设置的变量按错误处理 -v 显示Shell读入的命令语句 -x 显示执行的命令及其参数 - 关闭“-x”和“-v”标志 -- 重新设置位置参数, 或清除位置参数的设置
shift	把位置参数\$2, \$3, ……, \$N依次重新命名为\$1, \$2, ……, \$N-1, 原来的\$1和\$N不复存在, 位置参数总数相应地减1, 故新的 \$# 变量值也比原来的 \$# 变量值少1
shopt	<p>用于启用或禁用Shell控制选项的开关状态, 以规范Shell的行为。如果未带任何命令选项, 或使用了“-p”选项, 显示用户能够设置的所有Shell控制选项及其开关状态。shopt支持的部分重要命令选项如下:</p> <ul style="list-style-type: none"> -s 启用相应的控制选项 -u 禁用相应的控制选项 <p>利用shopt命令可以设置的部分重要Shell控制选项如下:</p> <ul style="list-style-type: none"> • cmdhist: 如果设置了这个Shell控制选项, bash将会尝试把一个多行的命令作为一个命令行保存到命令历史文件中, 以便用户能够容易地重新编辑多行命令。cmdhist的默认设置为on • histappend: 设置这个Shell控制选项之后, 当退出Shell时, bash将会把命令历史缓冲区中的命令历史记录附加到HISTFILE变量指定的命令历史文件中, 而不是覆盖这个文件。histappend的默认设置为off • huponexit: 如果把这个Shell控制选项设置为on, 当退出交互式的注册Shell时, bash将会向所有的作业发送一个SIGHUP信号。huponexit的默认设置为off • sourcepath: 如果设置了这个Shell控制选项, “.”或source内置命令将会使用PATH变量指定的目录检索指定的脚本文件。sourcepath的默认设置为on
source	类似于句点“.”命令, 用于读取指定的Shell脚本文件, 并在当前的Shell环境中运行, 运行结束后返回最后执行的一条命令的出口状态。如果文件名参数不是绝对路径, 使用PATH变量指定的目录检索脚本文件。在非posix环境中, 如果PATH变量指定的目录中不存在指定的文件, Bash将会在当前目录中检索给定的脚本文件。如果sourcepath控制选项处于禁用状态, 则禁止从PATH变量指定的目录中检索指定的文件
test	计算条件表达式。参见7.4节“test语句”
times	显示Shell或进程累计占用的用户和系统时间
trap	trap命令主要用于Shell脚本的信号捕捉与错误处理。当收到指定的信号时, Shell将会读入并执行指定的命令。如果指定的命令为null(“”), Shell将会忽略收到的每一个信号。在trap语句定义的处理动作结束之后, 其返回值将是调用trap之前执行的最后一个命令的出口状态。如果不希望在执行过程中被Ctrl-C等按键打断, 可以在Shell脚本中加入trap命令。详见第8章“Shell高级编程”
type	<p>type命令用于说明Shell如何解释指定的命令, 显示命令的完整路径名。例如:</p> <pre> \$ type find find is /usr/bin/find \$ type echo echo is a shell builtin \$ type ls ls is aliased to 'ls --color=auto' \$ </pre>
typeset	用于声明Shell变量或函数的类型属性。其功能及支持的选项完全等同于declare命令

内部命令	简单说明
ulimit	<p>用于设置或显示用户的资源限制。资源限制的值可以是一个数字，也可以是文字值unlimited。</p> <p>“-H”和“-S”选项分别用于指定硬性限制和软性限制。一旦设置，硬性限制值不能增加，但软性限制值可以增加，直至达到硬性限制值。如果“-H”或“-S”选项均未指定，则同时适用于软硬性限制。其他选项的意义说明如下（其中“-f”为默认的选项）：</p> <ul style="list-style-type: none"> -a 显示当前所有的资源限制 -c 显示内存映像转储文件（core）的最大容量限制（以1KB的数据块为单位） -d 显示进程可用数据区（段）的最大容量限制（以KB为单位） -e 列出可用的nice优先级调整值的最大值 -f 显示可以创建的最大文件的容量限制（以1KB的数据块为单位） -n 显示可用的文件描述符的最大数量限制 -s 显示可用的最大栈区的容量限制（以KB为单位） -t 显示每个进程可用的最大CPU时间量限制（以秒为单位） -u 显示单个用户可用的最大进程数量限制 -v 显示可用的虚拟内存的最大数量限制（以KB为单位）
umask	指定创建文件时应设定的默认访问权限。参见第5章“文件与目录操作”
unalias	撤销已经设定的命令别名。“-a”选项意味着撤销当前Shell运行环境中已经定义的所有命令别名
unset	用于清除Shell变量，把变量的值设为null。注意，这个命令并不影响位置参数
wait	<p>等待指定的作业结束，显示作业结束时的出口状态。如果未指定作业号，意味着等待当前作业及其所有子进程的结束。例如，“wait %1”意味着等待作业号为1的后台进程执行结束。</p> <p>“wait 6618”意味着等待进程ID为6618的后台作业完成</p>

7.3.2 部分命令介绍

1. “.”、source与exec命令

当Shell运行一个Shell脚本时，通常需要创建一个新的Shell进程，新的Shell进程将会继承其父Shell进程的环境变量（包括全局变量和已公布的变量），但不会继承未公布的本地变量。在Shell脚本中设置的变量，不管公布与否，只要退出Shell脚本，其中设置的变量将随之消失。为了利用Shell脚本设置应用程序可用的变量，通常需要使用“.”或source内部命令，在当前的Shell环境中解释执行Shell脚本。

与“.”或source命令类似，exec也是在当前进程的进程空间中执行指定命令的。但与“.”或source命令不同的是，“.”或source命令只能运行Shell脚本，而exec既可以运行Shell脚本，也可以运行编译的程序或命令。而且，在“.”或source命令调用的Shell脚本运行结束之后，将会返回调用点，但exec则不返回。此外，“.”或source命令运行的Shell脚本能够访问当前进程空间中的本地变量。

exec内部命令主要有两个用途：运行指定的命令时无需创建新的进程；重定向Shell脚本中的文件描述符。由于exec在运行指定的命令时并不创建新的进程，因而命令的执行速度更快。但是，由于exec命令并不返回到调用位置，因此只能用做Shell脚本中最后执行的一个命令。例如，在下列Shell脚本中，最后一个echo命令是不会执行的。

```
$ cat exec_demo
uname -n
```

```
exec date
echo "This line is never displayed."
$ exec_demo
iscas
2009年 12月 08日 星期二 01:01:58 CST
$
```

exec命令的第二个主要用途是重定向Shell脚本中的文件描述符，包括标准输入、标准输出和标准错误输出。如果Shell脚本中的某个位置出现下列命令，在随后读取标准输入时，其数据均取自exec命令指定的文件infile：

```
exec < infile
```

同样，下列exec命令将会把Shell脚本中随后命令的标准输出和标准错误输出分别重定向到指定的文件outfile和errfile：

```
exec > outfile 2> errfile
```

当以上述方式使用exec命令时，不会替代当前进程，exec命令之后仍然能够附加其他命令。

在使用exec命令重定向标准输出和标准错误输出之后，为了使用户能够看到Shell脚本中输出的任何提示信息，可以使用/dev/tty设备文件克服这一矛盾。/dev/tty是一个伪设备文件，表示用户当前使用的终端窗口（和键盘）。无论何时，都可以使用这个设备文件引用用户的终端窗口，而不必知道当前用户终端的实际设备文件名是什么（如果需要，可以使用tty命令获取用户终端的实际设备文件名）。

把Shell脚本中的标准输出重定向到/dev/tty，可以保证任何提示信息都能送达用户的终端窗口，而无需顾及用户究竟使用哪一个终端注册到Linux系统。另外，即使整个Shell脚本的标准输出和标准错误输出均重定向到某个文件，其间发送到/dev/tty的输出信息也不会受影响。

例如，下列三个echo命令分别把输出信息送到标准输出、标准错误输出和用户的终端窗口，当重定向整个Shell脚本的标准输出和标准错误输出时，发送到/dev/tty的输出信息仍然能够送达用户的终端窗口，而outfile和errfile则分别存有发送到标准输出和标准错误输出的信息：

```
$ cat exec_demo2
echo "message to standard output"
echo "message to standard error" 1>&2
echo "message to the user" > /dev/tty
$ exec_demo2 >outfile 2> errfile
message to the user
$ cat outfile
message to standard output
$ cat errfile
message to standard error
$
```

下列exec命令直接把Shell脚本的标准输出重定向到用户的终端窗口（实际上，这种重定向有点画蛇添足）：

```
exec > /dev/tty
```

如果把上述exec命令加到第一行，创建新的脚本文件exec_demo3，则由于已经把Shell脚本中的标准输出重定向到/dev/tty，此时，即使再把Shell脚本本身重定向到另一个文件，发送到标

准输出的信息仍然会输出到用户的终端窗口。但是，发送到标准错误输出的信息则不受影响。

```
$ cat exec_demo3
exec > /dev/tty
echo "message to standard output"
echo "message to standard error" 1>&2
echo "message to the user" > /dev/tty
$ exec_demo3 >outfile 2> errfile
message to standard output
message to the user
$ cat outfile
$ cat errfile
message to standard error
$
```

在Shell脚本中，采用exec命令重定向的一大优点是，不必在随后需要重定向的每个命令中——进行I/O重定向。

除了表示用户的终端窗口，/dev/tty也表示终端的键盘输入。如果在Shell脚本中使用下列exec命令，意味着读取用户终端的键盘输入（实际上，这种重定向也属画蛇添足）。之后，读取标准输入的所有命令均需接收来自终端键盘的输入。

```
exec < /dev/tty
```

2. “:”与true命令

实际上，“:”与true命令并不执行任何处理动作，其作用只是返回一个出口状态为0的测试条件。尽管两个命令的功能一样，但与“:”不同的是，true命令是Linux系统提供的一个外部命令。这两个命令经常用于实现不定循环，如用做while循环结构中的无限循环测试条件。显然，循环语句中还应增加一个break语句，以便当某个条件满足时能够退出循环。当然，也可以使用Ctrl-C键等方式强行中断循环语句的运行。

例如，当我们需要连续地观察一个日志文件或备份文件的大小是否按预期的那样不断增长时，可以使用下列命令，每5秒钟显示一次：

```
$ while true
> do
> ls -l backupfile
> sleep 5
> done
-rw-r--r-- 1 gqxing gqxing 11111 2009-11-7 17:18 backupfile
-rw-r--r-- 1 gqxing gqxing 22222 2009-11-7 17:18 backupfile
-rw-r--r-- 1 gqxing gqxing 33333 2009-11-7 17:18 backupfile
^C
$
```

3.echo命令

echo命令也许是Linux系统中应用最广泛的命令之一。echo命令主要用于显示各种信息，也可用于显示文件列表。作为显示信息的字符串，可以直接写在echo语句后面，也可以在字符串

前后加引号，例如：

```
$ echo Please enter your choice:
Please enter your choice:
$ echo *.c
atmcom.c atmmon.c atmstat.c handler.c listerner.c
$
```

利用“-e”选项，echo语句还支持若干转义字符，以便组织输出数据的显示格式。下面是部分常用的转义字符（详见第3章3.7节）：

- \n：表示在相应的位置插入一个换行符号；
- \t：表示在相应的位置插入一个制表符；
- \b：表示在相应的位置插入一个退格符号（Backspace）；
- \c：在输出所有的字符串参数之后，echo语句通常还会输出一个换行符。使用“\c”意味着取消输出换行符。此外，利用echo命令的“-n”选项也能够达到同样的目的。

下面的例子说明了怎样在一个echo命令中使用转义字符显示一个选择菜单。

```
$ echo -e "\n\t === Information Menu ===
\t1. CPU Model
\t2. Memory Configuration
\t3. End"

      === Information Menu ===
      1. CPU Model
      2. Memory Configuration
      3. End

$
```

4.read命令

read命令的主要功能是读来自标准输入的数据，然后存储到指定的变量参数中，实现交互式的变量赋值。read命令的语法格式简写如下（表7-7给出了read命令的部分选项及说明）：

```
read [-s] [-a aname] [-d delim] [-n nchars] [-p prompt] [-t timeout] [-u fd] [vars]
```

表7-7 read命令选项及其说明

选项	简单说明
-a aname	读取每一个字段（字或字符串），并依次赋予指定数组aname中的每一个元素
-d delim	使用指定的分隔符（而不是默认的换行符）作为输入数据的终止符
-n nchars	读取指定数量（nchars）的字符后立即返回。只要用户输入了nchars个字符，无需按下Enter键，read命令将会立即返回。如果输入的字符数量少于nchars，仅当按下Enter键，read命令才会响应。否则，read命令将一直处于等待状态
-p prompt	首先在标准错误输出中显示一个提示字符串（无换行，光标停留在提示字符串最后一个字符之后），然后等待用户输入。这个选项仅适用于从键盘中读取用户输入的数据
-s	禁止显示输入的任何字符，可用于输入密码数据
-t timeout	用于控制输入超时。如果在指定的时间（秒）内无法读取一个完整的输入行，将会引起read命令超时，并返回一个错误信息。如果不是从一个用户终端或管道读取输入数据，这个选项不起任何作用

选项	简单说明
<code>-u fd</code>	使用指定的整数fd作为文件描述符，以便read命令能够从指定的文件中读取数据。例如： <code>read -u 4 var1 var2</code> 等价于 <code>read var1 var2 <&4</code>

在实际应用中，为了在运行时直接从用户终端中读取数据，并为变量赋值，可在Shell脚本中增加下列read命令：

```
$ cat install
.....
echo "Enter a Serial number, then press <ENTER>."
read SNO
.....
$
```

如果read命令后面列有多个变量参数，输入的数据将按空格（IFS变量定义的字段分隔符，默认情况下为空格）分隔的单词顺序依次为每个变量赋值。如果输入的数据多于变量参数，多余的数据将会全部赋予最后一个变量参数：

```
$ cat readdata
echo -e "Please enter some data: \c"
read word1 word2 word3
echo "Word 1 is: $word1"
echo "Word 2 is: $word2"
echo "Word 3 is: $word3"
$ readdata
Please enter data: This is something
Word 1 is: This
Word 2 is: is
Word 3 is: something
$ readdata
Please enter data: This is something else
Word 1 is: This
Word 2 is: is
Word 3 is: something else
$
```

使用“-p”选项，可以省略echo语句，在read语句中直接增加提示信息。因此，上述代码的前两行可以合并为一个read语句：

```
read -p "Please enter some data: " word1 word2 word3
```

read命令的标准输入也可以重定向到一个文件。如果文件中含有多行数据，只有第一行数据会赋值到变量中。但是，如果利用循环语句，可以依次读取文件的每一行数据。如果成功地读取了任何输入数据，read命令将会返回一个值为0的出口状态。如果遇到文件结束标志，即读取了EOF（End Of File）标志字符，read命令将会返回一个非0值的出口状态。因此，可以使用read命令的这一特性作为循环结束的判断条件。下面是一个利用管道实现文件I/O重定向，从

中读取数据且为变量赋值的例子:

```
$ cat install2
#!/bin/bash
while read line
do
    set $line
    if [ "$3" = "swap" ]
    then
        .....
    fi
done < /etc/fstab
$
```

利用“-s”、“-t”以及“-n”等选项,还可以在执行read命令时禁止回显从键盘输入的数据,控制输入超时,以及读取指定数量的字符,而不必非按Enter键不可。例如,下列read命令用于读取单个字符:

```
$ read -s -nl -p "Hit a key " keypress
$ echo "Keypressed was \"${keypress}\"."
$
```

5.set与unset命令

set命令的一个重要用途是修改或重新设置位置参数的值。按照Shell的规定,用户不能直接为位置参数赋值。但是,利用set命令及其参数变量,则可以修改或重新设置位置参数的值。例如:

```
$ cat reset
echo $1 $2 $3 $4
set value1 value2
echo $1 $2 $3 $4
$ reset This is an argument
This is an argument
value1 value2
$ cat retain
echo $1 $2 $3 $4
set $1 value1 value2 $4
echo $1 $2 $3 $4
$ retain This is an argument
This is an argument
This value1 value2 argument
$
```

使用“set --”命令,也可以把参数变量的值赋予位置参数。如果“set --”命令后面未给定参数变量,意味着清除所有的位置参数。下面是一个重新分配位置参数的例子。

```
$ var="one two three four five"
$ set -- $var
$ echo $1 $2 $3 $4 $5
one two three four five
$ echo "$@"
one two three four five
```

```
$ set --                # 清除当前的所有位置参数
$ echo "$1"
$ echo "$@"
$
```

实际上，当使用给定的参数变量设置位置参数时，双减号“--”并非必需的，因而可以省略。下面是一个根据“uname -a”命令的输出，利用set命令设置位置参数的例子：

```
$ set `uname -a`
$ echo "Positional parameters after set `uname -a` :"
Positional parameters after set `uname -a` :
$ echo "Field 1 = $1"
Field 1 = Linux
$ echo "Field 2 = $2"
Field 2 = iscas
$ echo "Field 3 = $3"
Field 3 = 2.6.24-21-generic
$
```

如果不带任何选项或参数，set命令将会列出所有的环境变量、已经声明或设置的其他变量，以及函数定义等。这是set命令的另一个重要用途，例如：

```
$ set
.....
HOME=/home/gqxing
HOSTNAME=iscas
LANG=zh_CN.utf8
.....
$
```

unset命令用于清除Shell变量，把变量的值设置为null。注意，这个命令并不影响位置参数，例如：

```
$ variable=hello
$ echo "$variable"
hello
$ unset variable
$ echo "$variable"
$
```

6.set与shopt命令

Shell既是一个命令解释程序，提供用户与Linux系统的命令行界面，又是一个普通的系统命令，如不特别指定，Shell将会按照默认的环境设置运行。在Shell的运行过程中，如果需要，也可以使用set或shopt命令修改Shell的默认处理行为，定制自己的运行环境。为了启用Shell的某种功能特性，可以使用set命令的“-o”选项，而set命令的“+o”选项则用于关闭相应的功能特性。如果不加任何选项和参数，“set -o”命令将会显示set命令能够控制的每一个特性参数及其开关状态。例如，为了启用noclobber特性，防止重定向操作无意中覆盖同名的文件，可以使用下列命令：

```
$ set -o noclobber
$
```

为了关闭这一功能特性（默认），可以使用下列命令：

```
$ set +o noclobber
$
```

shopt是Bash中新增的一个内置命令。用于启用、关闭或显示Shell的部分功能特性。其中，“-s”选项用于启用指定的功能特性；“-u”选项用于关闭指定的功能特性。如果不加任何选项和参数，shopt命令将会显示shopt命令能够控制的每个功能特性及其开关状态。例如，采用下列shopt命令设置Shell的命令历史机制特性，能够把多行命令合并为一个命令项，在每个命令语句或关键字之间增加一个适当的分号“;”分隔符。

```
$ shopt -s cmdhist
$
```

为了关闭这一功能特性，可以使用下列shopt命令：

```
$ shopt -u cmdhist
$
```

为了查询某一功能特性的开关状态，可以使用下列shopt命令：

```
$ shopt cmdhist
cmdhist      on
$
```

7.expr命令

expr命令用于计算表达式的值，然后把计算结果送到标准输出。其中的表达式可以是字符串比较表达式、整数算术运算表达式或模式匹配表达式。其语法格式简写如下：

```
expr expression
```

表7-8给出了expr命令支持的各种字符串比较表达式，以及计算结果的说明。expr命令基本上已被test命令或“[[...]]”结构所取代，且在“[[...]]”结构中，“>”和“<”符号之前也无需加转义符号，故现在很少使用expr命令做字符串比较（但是，test命令和“[[...]]”结构不支持“>=”和“<=”比较）。

表7-8 expr命令支持的字符串比较表达式

表达式	计算结果
<i>str1</i> = <i>str2</i>	如果字符串str1等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
<i>str1</i> \> <i>str2</i>	如果字符串str1大于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
<i>str1</i> \< <i>str2</i>	如果字符串str1小于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
<i>str1</i> \>= <i>str2</i>	如果字符串str1大于等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1

（续表）

表达式	计算结果
<i>str1</i> \<= <i>str2</i>	如果字符串str1小于等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1
<i>str1</i> != <i>str2</i>	如果字符串str1不等于str2，则计算结果为真，同时输出1，但返回值为0。反之，如果计算结果为假，则输出0，但返回值为1

下面的例子说明了怎样利用expr命令测试字符串表达式:

```
$ TIMEZONE=CST
$ expr "$TIMEZONE" = "CST"
1
$ echo $?
0
$ expr "$TIMEZONE" = "GMT"
0
$ echo $?
1
$
```

expr命令也可用于计算整数表达式的值，计算结果会被发送到标准输出。因此，如果想把计算结果值保存到一个变量中，需要采用命令替换的方式实现。

表7-9给出了expr命令支持的各种整数算术运算表达式，以及有关计算结果的说明。expr命令仅支持整数表达式的计算，如果给定的参数不是整数，expr命令将会输出一个出错信息。

表7-9 expr命令支持的整数算术运算表达式

表达式	简单说明
<i>exp1</i> + <i>exp2</i>	计算整数表达式exp1与exp2的和
<i>exp1</i> - <i>exp2</i>	计算整数表达式exp1与exp2的差
<i>exp1</i> * <i>exp2</i>	计算整数表达式exp1与exp2的乘积。注意，星号前应加转义符号“\”
<i>exp1</i> / <i>exp2</i>	计算整数表达式exp1与exp2的商
<i>exp1</i> % <i>exp2</i>	计算整数表达式exp1与exp2的余数

下面的例子说明了怎样利用expr命令计算整数表达式，以及怎样利用命令替换的方式保存expr命令的计算结果。

```
$ n=3
$ expr $n + 7
10
$ n=`expr $n + 1`
$ echo $n
4
$
```

8.let命令与“((...))”结构

let命令和双括号“((...))”结构用于计算和测试整数算术表达式，执行整数算术运算。实际

上, let命令和 “((...))” 结构是对expr命令的简化, 取代并扩展了expr命令的整数算术运算功能。

除了上述5种算术运算之外, let命令和 “((...))” 结构还支持+=、-=、*=、/=和%=等运算符。此外, 在let语句中, 表示乘法运算的符号 “*” 之前无需增加转义符号。表7-10给出了let命令和 “((...))” 结构支持的各种整数算术运算, 及其简单说明。

表7-10 let命令和 “((...))” 结构支持的算术运算

运算符	简单说明
<i>exp1</i> + <i>exp2</i>	计算整数表达式 <i>exp1</i> 与 <i>exp2</i> 的和
<i>exp1</i> - <i>exp2</i>	计算整数表达式 <i>exp1</i> 与 <i>exp2</i> 的差
<i>exp1</i> * <i>exp2</i>	计算整数表达式 <i>exp1</i> 与 <i>exp2</i> 的乘积
<i>exp1</i> / <i>exp2</i>	计算整数表达式 <i>exp1</i> 与 <i>exp2</i> 的商
<i>exp1</i> % <i>exp2</i>	计算整数表达式 <i>exp1</i> 与 <i>exp2</i> 的余数。可用于生成位于某个范围内的数字
<i>var</i> += <i>exp</i>	组合加运算符。把表达式的值加到变量中。例如, 执行let "var += 5"之后, 变量var的值将增加5
<i>var</i> -= <i>exp</i>	组合减运算符。从变量中减去表达式的值。例如, 执行let "var -= 3"之后, 变量var的值将减少3
<i>var</i> *= <i>exp</i>	组合乘运算符。计算变量与表达式的乘积, 再把结果赋值到变量中。例如, 执行let "var *= 4"之后, 变量var的值将扩大4倍
<i>var</i> /= <i>exp</i>	组合除运算符。把变量除以表达式后的商再赋值到变量中。例如, 执行let "var /= 2"之后, 变量var的值将缩小2倍
<i>var</i> %= <i>exp</i>	组合模运算符。把变量除以表达式后的余数再赋值到变量中

下面以不同的运算符说明怎样利用let命令计算整数表达式。

```
$ n=1
$ let "n = $n + 1"
$ echo "$n"
2
$ let "n = n + 1"
$ echo "$n"
3
$ let n=$n+1
$ echo "$n"
4
$ let n=n+1
$ echo "$n"
5
$ (( n = n + 1 ))
$ echo "$n"
6
$ let "n += 1"
$ echo "$n"
7
$ let n+=1
$ echo "$n"
8
$
```

let命令和((...))结构也可以返回算术表达式计算结果的出口状态。如果算术表达式的计算结果为0，出口状态为1。如果计算结果为非0值，出口状态为0。这一点与test语句、“[...]”和“[[...]]”结构恰好相反。但其比较运算的判断逻辑是一致的。参见下面的例子：

```
$ (( 5-5 ))
$ echo "The exit status is $?"
The exit status is 1
$ (( 5+5 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5/5 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5 > 4 ))
$ echo "The exit status is $?"
The exit status is 0
$ (( 5 > 9 ))
$ echo "The exit status is $?"
The exit status is 1
$ let "1<2"
$ echo "The exit status is $?"
The exit status is 0
$ let "5<2"
$ echo "The exit status is $?"
The exit status is 1
$
```



在let语句中，如果未加双引号，表达式之间不能有空格。此外，为了加快Shell的运算速度，提高Shell编程的严谨性，最好事先利用typeset命令定义变量的类型，并对变量进行初始化。例如，下面的例子首先利用typeset命令把变量z定义为整数变量，同时初始化为0。然后采用let命令和“((...))”结构进行计算。

```
$ typeset -i z=0
$ let z=z+1
$ let "z += 1"
$ z=$((z+1))
$ z=$((z+1))
$ echo $z
4
$
```

9. 数值常数

Shell脚本按十进制数值解释字符串中的数字字符，除非数字前有特殊的前缀或记号。如果数字前面有一个数值零(0)，表示这是一个八进制的数，0x或0X表示一个十六进制的数。BASE#NUMBER表示以BASE(2~64)为底数，以NUMBER为指数的数值。

```
$ let "dec = 32" # 默认的十进制数值
$ echo "decimal number = $dec"
decimal number = 32
$ let "oct = 032" # 八进制数值
```



```

$ echo "octal number = $oct"
octal number = 26
$ let "hex = 0x32"                                # 十六进制数值
$ echo "hexadecimal number = $hex"
hexadecimal number = 50
$ let "bin = 2#111100111001101"                    # 二进制数值
$ echo "binary number = $bin"
binary number = 31181
$

```

7.3.3 命令替换

命令替换的目的是获取命令的输出，为变量赋值，或对命令的输出做进一步的处理。命令替换的实现有两种方法：一是使用反向引号“```”引用命令；二是采用“`$(...)`”形式引用命令。其语法格式简写如下：

```
`command`
```

或

```
$(command)
```

例如，为了获取date命令的输出，并把输出结果赋予today变量，可以使用下列命令替换形式：

```

$ today=`date`
$ echo $today
2009年 12月 08日 星期二 10:49:36 CST
$

```

下面的例子是利用第二种命令替换形式获取uname命令输出系统的主机名，并把结果赋予machine变量：

```

$ machine=$(uname -n)
$ echo $machine
iscas
$

```

原则上，命令替换中的命令可以是任何命令，包括外部命令、Shell脚本，甚至是Shell脚本中定义的函数。严格地讲，命令替换实际上蕴含着两个过程：执行相应的命令，获取命令标准输出中的数据。在获取命令的输出数据之后，可以使用赋值运算符（=），把数据赋予某个变量，或做进一步的处理。

命令的输出也可以用做另一个命令的输入参数，甚至可用于生成for循环中的参数表（参见第8章“Shell高级编程”）。例如，如果文件filename中包含需要删除的文件列表，使用下列命令可以删除文件filename中指定的所有文件：

```

$ rm -rf `cat filename`
$

```



注意

在执行上述命令时，如果出现类似于“arg list too long”的出错信息，最好改用“`xargs rm -<filename`”命令。

不管采用哪一种命令替换形式，其最终效果是完全一样的。下面两个例子是分别采用两种命令替换形式的输出结果：

```
$ cfile=`ls *.c`
$ echo $cfile
atmcom.c atmmon.c atmstat.c handler.c listerner.c
$ cfile2=$(ls *.c)
$ echo $cfile2
atmcom.c atmmon.c atmstat.c handler.c listerner.c
$
```



在使用echo命令输出一个未加引号引用的变量，而相应的变量又是采取命令替换的形式赋值时，将会抛弃命令输出数据后面的换行符，从而有可能导致不期望的结果。在下列例子中，我们希望显示一个文件列表，但实际的输出数据却是另外一种结果：

```
$ dir_listing=`ls -l /etc/kernel`
$ echo $dir_listing          # 未使用引号
total 8 drwxr-xr-x 2 root root 4096 Nov 15 00:11 header_postinst.d drwxr-xr-x
2 root root 4096 Nov 15 00:11 postinst.d
$
```

这显然不是我们预期的输出结果。如果在引用变量时增加一对双引号，其输出内容恰是我们希望看到的结果：

```
$ echo "$dir_listing"        # 使用引号
total 8
drwxr-xr-x 2 root root 4096 Nov 15 00:11 header_postinst.d
drwxr-xr-x 2 root root 4096 Nov 15 00:11 postinst.d
$
```

在命令替换中，使用I/O重定向的方式或cat命令，还可以把文件的全部内容赋予一个变量。但在实际的应用过程中，把一个较大文本文件的内容赋予一个变量并非必要。尤其不应当把一个二进制文件的内容赋予变量。例如，如果确实需要，我们可以使用下列命令把文件/etc/timezone的内容赋予变量var1：

```
$ cat /etc/timezone
Asia/Shanghai
$ var1=`cat /etc/timezone`
$ echo $var1
Asia/Shanghai
$
```

如果稍做变换，可以把上述命令替换改写如下（实际上，可以把“<fname”看做“cat fname”命令的一个特例）：

```
$ var2=`< /etc/timezone`
$ echo $var2
Asia/Shanghai
$
```



注意 采用命令替换的形式为变量赋值时，变量中可能包含空格，甚至包含控制字符。

7.4 test语句

每一种编程语言都具有条件测试功能，而条件测试的结果将决定程序的控制流向和下一步的处理动作。通过test命令及其简化形式（两种方括号测试结构），Shell编程也支持这一功能。test语句与if/then和case结构语句一起，构成了Shell编程的控制转移结构，与while和until等结构语句一起，构成了Shell编程的循环结构。

通常，Shell执行的test命令是Shell自己提供的内部命令。在执行test语句时，如果不特别指定，Shell不会调用外部的系统命令/usr/bin/test。

test命令（及其简化形式）的主要功能是计算随后的表达式，如检查文件的属性，比较字符串，或比较字符串表示的整数值等，然后以表达式的计算结果作为test命令的出口状态。如果测试条件为真，test命令将会返回0，否则返回一个非0数值。

test命令经常用于控制Shell脚本的执行流向。test语句中的选项与参数或表达式描述了测试的条件。测试条件可以是文件属性检查、字符串比较或整数值比较。在计算test语句中的表达式之前，Shell首先执行变量替换或命令替换，然后再执行条件测试。

在Bash中，test命令的语法格式主要有下列三种形式：

- test expression
- [expression]
- [[expression]]

“[...]”是一种简化的、效率更高的test命令。而“[[...]]”结构则是一种比 “[...]”结构更通用的测试结构，也是扩展的test命令。同test命令一样，这种测试结构用于计算括号内的表达式，测试文件的属性，或比较字符串及其相应的整数值。然后再根据计算和测试的结果，返回相应的出口状态（0或1）。注意，方括号内侧的两边必须各加一个空格。

例如，在删除文件时，为了避免出错，可以先测试文件是否存在。如果文件确实存在，然后再执行文件删除操作，如：

```
$ fname=/tmp/somefile
$ if test -e $fname
> then
> rm -f $fname
> echo "File $fname has removed."
> fi
File /tmp/somefile has removed.
$
```

采用 “[...]” 测试结构，可把上述代码片段改写如下：

```
$ fname=/tmp/somefile
$ if [ -e $fname ]
> then
> rm -f $fname
> echo "File $fname has removed."
```

```
> fi
File /tmp/somefile has removed.
$
```

在Shell脚本中，使用 “[[...]]” 测试结构，而不用 “[...]” 结构，有时能够避免出现逻辑错误。例如，在 “[[...]]” 测试结构中，可以使用 “&&”、“||”、“<” 和 “>” 等运算符。但如果在 “[...]” 测试结构中使用上述运算符就会出错。示例如下：

```
$ fname=/tmp/somefile
$ flag=yes
$ if [[ -e $fname && "$flag" = "yes" ]]
> then
> rm -f $fname
> echo "File $fname has removed."
> fi
File /tmp/somefile has removed.
$
```



“[[...]]” 测试结构不允许执行文件名生成和单字解析操作。

7.4.1 文件测试运算符

文件测试主要指文件的状态和属性测试，如文件是否存在、文件的类型、文件的访问权限以及其他属性等。

文件的访问权限分为三种类型：文件属主、同组用户和其他用户。如果Shell脚本的执行者是文件的属主，则检查相应于文件属主的访问权限；如果执行者不是文件属主，而是与文件属主同组的成员，则检查相应于同组用户的访问权限；如果执行者既非属主，又非同组成员，则检查相应于其他用户的访问权限。

在test语句中，文件名参数必须显式地指定，不能为空。文件名参数可以是实际的文件名，也可以是变量替换、命令替换或文件名生成机制等的最终结果。如果文件名参数是由替换方式生成的，必须使用双引号括起来。

表7-11是test语句中常用的部分文件属性测试表达式。

表7-11 文件属性测试表达式

表达式	简单说明
-e file	如果给定的文件存在，则条件测试的结果为真
-r file	如果给定的文件存在，且其访问权限是当前用户可读的，则条件测试的结果为真
-w file	如果给定的文件存在，且其访问权限是当前用户可写的，则条件测试的结果为真
-x file	如果给定的文件存在，且其访问权限是当前用户可执行的，则条件测试的结果为真
-s file	如果给定的文件存在，且其大小大于0，则条件测试的结果为真
-f file	如果给定的文件存在，且是一个普通文件（非目录或设备文件），则条件测试的结果为真
-d file	如果给定的文件存在，且是一个目录，则条件测试的结果为真
-L file	如果给定的文件存在，且是一个符号连接，则条件测试的结果为真

(续表)

表达式	简单说明
<code>-c file</code>	如果给定的文件存在, 且是字符特殊文件(如终端等), 则条件测试的结果为真
<code>-b file</code>	如果给定的文件存在, 且是块特殊文件(如磁盘等), 则条件测试的结果为真
<code>-p file</code>	如果给定的文件存在, 且是命名的管道文件, 则条件测试的结果为真
<code>-u file</code>	如果给定的文件存在, 且其setuid标志位已经设置, 则条件测试的结果为真。如果文件属主为超级用户的可执行文件已经设置了setuid标志, 即使是普通用户, 在执行期间也具有超级用户的特权。对于某些可执行文件而言, 这是非常有用的。例如, passwd命令需要更新系统文件/etc/shadow。如果没有设置setuid标志位, 普通用户就无法使用passwd命令修改密码。为了了解一个文件是否设置了setuid标志位, 可以使用下列命令显示相应的文件, 检查其文件属主访问权限字段是否存在一个字符“s”:
	<pre>\$ ls -l /usr/bin/passwd -rwsr-xr-x 1 root root 41292 Jul 31 21:55 /usr/bin/passwd \$</pre> <p>注意, 设置setuid标志位可能会留下安全隐患。但对Shell脚本来讲, setuid标志位没有影响</p>
<code>-g file</code>	如果给定的文件存在, 且其setgid标志位已经设置, 则条件测试结果为真。如果某个目录设置了setgid标志位, 则在该目录中创建的文件属于同组成员。这对于工作组成员共享同一目录是非常有用的
<code>-k file</code>	如果给定的文件存在, 且其粘性标志位已经设置, 则条件测试结果为真。粘性标志位是一种特殊的文件访问权限。如果某个可执行文件设置了粘性标志位, 当调度相应的程序文件执行时, 在整个运行过程中, 相应的程序将始终保持在高速缓存中, 访问和执行的速度因而更快捷。如果设置的是目录, 将限制用户的访问权限。如果一个目录设置了粘性标志位, 在使用下列命令显示相应的文件或目录时, 其访问权限字段将会出现一个字符“t”:
	<pre>\$ ls -ld /tmp drwxrwxrwt 16 root root 4096 Dec 16 17:41 /tmp \$</pre> <p>如果某个共享目录已经设置了粘性标志位, 且任何用户均具有写的访问权限, 则用户只能删除目录中属于自己的文件。这将防止用户无意中覆盖或删除共享目录下其他用户拥有的文件。注意, 在Linux系统中, 粘性标志位仅适用于目录</p>
<code>f1 -nt f2</code>	如果给定的文件f1存在, 且其修改日期比文件f2新, 则条件测试的结果为真
<code>f1 -ot f2</code>	如果给定的文件f1存在, 且其修改日期比文件f2早, 则条件测试的结果为真
<code>f1 -ef f2</code>	如果给定的文件f1和f2存在且指向的是同一个物理文件, 则条件测试的结果为真
<code>!</code>	逻辑非运算符。当与上述表达式一起使用时, 测试结果与其本意恰好相反

文件测试主要用于Shell脚本, 许多守护进程在启动过程中都会采用文件测试表达式检测某个文件是否存在, 以便决定是否运行相应的命令或Shell脚本。下面是一个取自/etc/init.d/ssh启动脚本的代码片段, 前者测试环境变量的设置文件是否存在, 如果存在, 则运行相应的脚本文件, 以设置默认的运行环境; 后者检查必要的目录是否存在, 如果不存在, 则建立相应的目录:

```
$ cat /etc/init.d/ssh
.....
if test -f /etc/default/ssh; then
    . /etc/default/ssh
fi
```

```
.....
if [ ! -d /var/run/sshd ]; then
    mkdir /var/run/sshd
    chmod 0755 /var/run/sshd
fi
.....
$
```

7.4.2 字符串测试运算符

表7-12给出了test语句中常用的字符串测试表达式。

表7-12 字符串测试表达式

表达式	简单说明
-z <i>str</i>	如果给定字符串的长度为0，则条件测试的结果为真。在“! -z”情况下，如果字符串未加引号，或在test语句中单独使用未加引号的字符串，结果将是不可靠的
-n <i>str</i>	如果给定字符串的长度大于0，则条件测试的结果为真。“-n”测试要求字符串前后必须加引号
<i>s1</i> = <i>s2</i>	如果给定字符串s1等同于字符串s2，则条件测试的结果为真
<i>s1</i> != <i>s2</i>	如果给定字符串s1不等同于字符串s2，则条件测试的结果为真
<i>s1</i> < <i>s2</i>	基于字符的ASCII编码值，如果给定的字符串s1小于字符串s2，则条件测试的结果为真。其部分用法列举如下： • test <i>s1</i> < <i>s2</i> • [<i>s1</i> \< <i>s2</i>] • [[<i>s1</i> < <i>s2</i>]] 注意，在单方括号 “[...]” 中，小于号 “<” 前需加转义符号
<i>s1</i> > <i>s2</i>	基于字符的ASCII编码值，如果给定的字符串s1大于字符串s2，则条件测试的结果为真。其部分用法列举如下： • test <i>s1</i> > <i>s2</i> • [<i>s1</i> \> <i>s2</i>] • [[<i>s1</i> > <i>s2</i>]] 注意，在单方括号 “[...]” 中，大于号 “>” 前需加转义符号

1. 字符串等同性测试

test语句支持两种字符串等同性比较测试：等于（=）测试用于比较两个表达式的相等性；不等（!=）测试用于比较两个表达式的不等性。

下面是两个test语句的例子。第一个例子测试两个字符串的相等性；第二个例子测试两个字符串的不等性。注意，比较运算符两端必须各有至少一个空格。如果漏掉了空格，test语句就会把比较运算符看做赋值运算符或待测试字符串的一部分：

```
$ name="John"
$ test "$name" = John
$ echo $?
0
$ name="John  "
$ [ "$name" = John ]
```

```
$ echo $?
1
$
```

此外还要注意，在test语句的字符串比较表达式中，引用的变量或字符串前后一定要加双引号。否则，当变量或字符串表达式的值为null时，Shell将会忽略变量或字符串表达式的存在，导致语法错误。例如，当NOTSET变量未设置时，第一个test命令认为“!=”运算符左边的参数不存在，故测试出错：

```
$ echo ${NOTSET}
$ test ${NOTSET} != "hello"
-bash: test: !=: unary operator expected
$ test "${NOTSET}" != "hello"
$ echo $?
0
$
```

在上述例子中，第一个test语句执行有误说明漏掉了双引号。Shell按IFS处理机制从命令行中删除值为null的字符串参数，因此，当执行test语句时，Shell只看到两个参数：即“!=”与“hello”，因而认为不满足字符串比较所需的三个参数。

2. 字符串长度测试

test语句也可用于测试字符串表达式的长度。test语句中的“-z”和“-n”选项分别用于测试字符串表达式的长度是否为0或非0。如果未明显地指定任何选项，“-n”将会成为默认的选项。也就是说，如果test语句中只有一个参数，则当字符串参数包含一个或多个字符时，测试条件的结果为真，因而返回一个0值。例如：

```
$ test -z "string"
$ echo $?
1
$ test -n "string"
$ echo $?
0
$ test "string"
$ echo $?
0
$
```

字符串长度测试经常用于安装软件时检查变量是否已经设置，根据变量或形式参数的设置与否，组织控制结构，决定Shell脚本命令执行的控制流向。

7.4.3 整数测试运算符

test语句还可用于比较字符串表达式中包含的整数值。整数字符串中不能包含任何非数字字符，但前后可以有空格字符。下面是一些合法的整数字符串的例子（Bourne Shell还允许数字后面包含其他非数字字符）：

"486"	486
"15 "	15
" 123"	123

表7-13给出了test语句中常用的整数测试表达式。

表7-13 整数测试表达式

表达式	简单说明
<i>exp1</i> -eq <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值等于表达式 <i>exp2</i> 的整数值，则计算结果为真
<i>exp1</i> -ne <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值不等于表达式 <i>exp2</i> 的整数值，则计算结果为真
<i>exp1</i> -gt <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值大于表达式 <i>exp2</i> 的整数值，则计算结果为真
<i>exp1</i> -lt <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值小于表达式 <i>exp2</i> 的整数值，则计算结果为真
<i>exp1</i> -ge <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值大于等于表达式 <i>exp2</i> 的整数值，则计算结果为真
<i>exp1</i> -le <i>exp2</i>	如果表达式 <i>exp1</i> 的整数值小于等于表达式 <i>exp2</i> 的整数值，则计算结果为真

在test语句中，整数值的比较采用C语言中的atoi()函数，把字符串转换成等价的ASCII整数值。下列两个例子说明了怎样使用test语句比较整数字符串表达式：

```
$ test "123" -eq "123"
$ echo $?
0
$ test "123" -eq " 123 "
$ echo $?
0
$
```

7.4.4 逻辑运算符

test语句支持表达式的逻辑运算。其中，符号“!”表示逻辑非运算；符号“-a”或“&&”表示逻辑与运算；符号“-o”或“|”表示逻辑或运算，如表7-14所示。

表7-14 逻辑运算表达式

表达式	简单说明
(<i>exp</i>)	用于计算括号中的组合表达式。如果整个表达式的计算结果为真，则测试结果也为真
! <i>exp</i>	对表达式进行逻辑非运算，即对测试结果求反。如果表达式的计算结果为假，则最终的测试结果为真
<i>exp1</i> -a <i>exp2</i> <i>exp1</i> && <i>exp2</i>	对两个表达式进行逻辑与运算。如果两个表达式的计算结果均为真，最终的测试结果才为真。注意，单方括号 ([...]) 结构中不允许使用&&运算符。逻辑与运算符的部分用法列举如下： • test <i>exp1</i> -a <i>exp2</i> • [<i>exp1</i> -a <i>exp2</i>] • [<i>exp1</i>] && [<i>exp2</i>] • [[<i>exp1</i> && <i>exp2</i>]]
<i>exp1</i> -o <i>exp2</i> <i>exp1</i> <i>exp2</i>	对两个表达式进行逻辑或运算。只要两个表达式的计算结果中有一个为真，最终的计算结果就为真。同样，单个方括号 “[...]” 结构中不允许使用 “ ” 运算符。逻辑或运算符的部分用法列举如下： • test <i>exp1</i> -o <i>exp2</i> • [<i>exp1</i> -o <i>exp2</i>] • [<i>exp1</i>] [<i>exp2</i>] • [[<i>exp1</i> <i>exp2</i>]]

在比较两个文件时, 可以使用diff命令。比较之前, 为了确保两个文件都存在, 可以写出下列Shell代码片段:

```
if [ -f file1 -a -f file2 ]
then
diff file1 file2
fi
```

下面的例子说明了怎样使用各种逻辑运算符:

```
$ test ! -d /tmp
$ echo $?
1
$ test -d /tmp -a -x /tmp
$ echo $?
0
$ ! [ -d /tmp -a -x /tmp ]
$ echo $?
1
$ [[ -d /tmp && -x /tmp ]]
$ echo $?
0
$
```

7.5 命令行的解释执行过程

Shell命令行的解释执行是一个复杂的处理过程, 了解命令行的解释执行过程有助于用户正确地访问系统, 准确地输入命令, 避免误用或出现意外, 影响命令的最终运行结果。例如, 假定我们想用一個变量的值实现I/O重定向, 把命令的输出保存到一个文件中, 其结果如下:

```
$ SAVEFILE="> /tmp/savefile"
$ echo something $SAVEFILE
something > /tmp/savefile
$ cat /tmp/savefile
cat: /tmp/savefile: No such file or directory
$
```

从上述命令的运行结果可以看出, Shell并没有把echo命令的输出重定向到文件savefile中, 而是把“something”和SAVEFILE变量的值“>/tmp/savefile”一起输出到终端窗口(标准输出)上。实际上, Shell的I/O重定向是在变量替换之前执行的。替换SAVEFILE变量之后, Shell只是把变量的值作为echo命令参数的一部分送到标准输出, 因而没有创建预期的文件/tmp/savefile。

因此, 有必要在此讨论一下Shell命令行的解释执行过程。Shell命令行的解释执行过程大体可以分为16个步骤。按照顺序列举如下:

- (1) 读取命令行;
- (2) 命令历史替换;
- (3) 命令别名替换;
- (4) 花括号扩展;
- (5) 波浪号替换;

- (6) I/O重定向;
- (7) 变量替换;
- (8) 算术运算结果替换;
- (9) 命令替换;
- (10) 单词解析;
- (11) 文件名生成;
- (12) 引用字符处理;
- (13) 进程替换;
- (14) 环境处理;
- (15) 执行命令;
- (16) 跟踪执行过程。

7.5.1 读取命令行

Shell命令行解释执行过程的第一步是读取命令行。命令行可以来自终端，也可以取自一个普通的文本文件，如Shell脚本文件。

不管是以交互方式访问Shell，还是运行Shell脚本，Shell都需要读取命令行。如果是交互式Shell，Shell还需要在用户终端上回显其读取的每一个字符，然后对读取的命令行进行解析和处理。在解析和处理每个命令行之前，Shell需要读取完整的命令行。部分Shell内置命令，如if/then、for和case等结构语句，函数定义与长字符串等，通常都可能需要跨越多个物理行。当读取多行形式的命令语句时，在进一步解析和处理之前，Shell将会连续地读取整个命令语句。在交互会话期间，Shell会采用辅助命令提示符“>”（PS2变量的默认值）的形式，提示用户输入后续的命令语句，此时用户可以继续输入，直至Shell认为命令行的输入已经结束。例如：

```
$ echo "***** NOTICE *****"
> Today is Christmas.
> Every body could go home after 12:00."
***** NOTICE *****
Today is Christmas.
Every body could go home after 12:00.
$
```

在读取命令行时，Shell将会逐个判断读取的每一个字符，直至遇到一个分号“;”、后台进程符号“&”、逻辑与“&&”、逻辑或“||”或换行字符时，整个命令行的读取过程才算结束。如果读取的是一个结构语句，如if/then、for或while等，Shell将会完整地读入整个结构语句。

在读入一个完整的命令或结构语句之后，Shell开始对命令语句进行语法分析，把命令语句分解为一系列单词或关键字。通常，Shell假定每个单词或关键字都是由空格或制表符分隔的一系列连续字符组成的（参见IFS变量的说明）。Shell从命令语句行首的第一个字符开始解析，直至行尾结束，依次剥离出一系列单词，包括由“<”、“>”、“|”和“'”等特殊字符组成的单字。不管IFS变量的设置如何，这一解析过程总是如此处理。至于如何处理IFS变量，则是另外一个解析步骤。

7.5.2 命令历史替换

在读取一个完整的命令行之后，Shell首先会检查是否需要使用命令历史缓冲区中记录的命令替换读取的命令，是否需要编辑命令。如果需要，则取出相应的命令，再经过适当的校正之后，使之作为当前需要执行的命令。例如，当输入“!!”命令时，意味着重新执行先前执行的最后一条命令，Shell于是会从命令历史记录中取出相应的命令，替换“!!”命令。

通常，交互式Shell总是会首先执行命令历史机制的命令替换处理过程。如果想要提高Shell命令行解释执行的效率，省略这一处理步骤，可以使用下列命令禁止Shell执行命令历史替换（注意，命令历史替换处理不适用于非交互式Shell（如运行Shell脚本））：

```
$ set +o histexpand
$
```

7.5.3 别名替换

命令行解释执行的第二步是别名替换。在此步骤中，Shell将检查读取的命令是否为命令别名。如果确为命令别名，则根据定义，使用原始的命令予以替换。通常，交互式Shell总是会执行命令别名检查，非交互式Shell则禁用这一功能特性。为了省略这一处理步骤，可以使用下列命令禁止Shell执行命令别名替换：

```
$ shopt -u expand_aliases
$
```

7.5.4 花括号扩展

花括号提供了一种简单的文件名生成方法。下面的例子说明了怎样利用花括号扩展机制指定文件名，在/home/gqxing/scripts目录中一次创建4个子目录

```
$ mkdir ~/scripts/{old,new,tools,admin}
$
```

在上述例子中，Shell尝试把花括号中以逗号“,”分隔的一系列单词与花括号之外的字符串连接在一起，生成若干由空格分隔的字符串，用于指定或匹配当前目录中的文件。

当文件的路径名较长时，花括号扩展机制是非常有用的。例如，为了把/home/gqxing/src目录中的部分C程序（main.c、list.c、scan.c和mon.c）复制到当前目录，可以使用下列命令：

```
$ cp /home/gqxing/src/{main,list,scan,mon}.c .
$
```

但是，Shell并不总是尝试使用花括号扩展机制匹配现有的文件名的，实际上，也可以使用花括号扩展机制生成任何字符串。例如，利用下列命令生成的字符串并非实际存在的目录文件名：

```
$ echo chap-{one,two,three,four}
chap-one chap-two chap-three chap-four
$
```

通常，交互式Shell总是启用花括号扩展机制，非交互式Shell则禁用花括号扩展机制。为了禁止Shell使用花括号扩展机制，可以输入下列命令：

```
$ set +o braceexpand
$
```

7.5.5 波浪号替换

为了指定用户的主目录，除了使用\$HOME变量之外，还可以在用户名之前加一个波浪号“~”，或直接使用波浪号表示当前用户的主目录。当命令行中出现波浪号“~”这一特殊字符时，Shell将会继续读取随后的字符串，直至遇到一个斜线字符“/”或空白字符（表示单字的结束）为止，然后验证读取的字符串是否为一个有效的注册用户名。如果读取的字符串为空（即后面只有一个斜线字符“/”，因而只有波浪号“~”本身），Shell将会使用当前用户的HOME变量值替换波浪号“~”。例如：

```
$ echo $HOME
/home/gqxing
$ echo ~
/home/gqxing
$ ls -l ~/src
total 128
-rw-r--r-- 1 gqxing gqxing 18222 Nov  8 11:50 atmcom.c
-rw-r--r-- 1 gqxing gqxing 11802 Nov  8 11:52 atmmon.c
-rw-r--r-- 1 gqxing gqxing 23221 Nov  8 11:53 atmstat.c
-rw-r--r-- 1 gqxing gqxing 11742 Nov  8 11:53 handler.c
-rw-r--r-- 1 gqxing gqxing 55596 Nov  8 11:56 listener.c
$
```

如果波浪号“~”之后是一个有效的注册用户名，Shell将以相应用户主目录的路径名替换波浪号“~”和用户名。如果“~”之后既不为空，也不是一个有效的用户名，Shell不会做任何替换。例如：

```
$ echo $HOME
/home/gqxing
$ ls -l ~gqxing
total 20
drwxr-xr-x 2 gqxing gqxing 4096 Nov  8 09:48 conf
drwxr-xr-x 2 gqxing gqxing 4096 Nov  8 15:16 incl
drwxr-xr-x 2 gqxing gqxing 4096 Nov  8 09:57 lib
-rw-r--r-- 1 gqxing gqxing 1020 Nov  8 17:51 makefile
drwxr-xr-x 2 gqxing gqxing 4096 Nov  8 18:37 src
$ echo ~xxx
~xxx
$
```

此外，如果波浪符“~”之后为加号“+”或减号“-”，则“~+”表示PWD变量的值，即当前工作目录；“~-”表示OLDPWD变量的值，即先前的工作目录。例如：

```
$ cd /etc
$ cd
$ echo $PWD
/home/gqxing
$ echo ~+
```

```

/home/gqxing
$ echo $OLDPWD
/etc
$ echo ~-
/etc
$

```

7.5.6 I/O重定向

如果不了解文件描述符、进程文件表（也称文件描述符表或用户文件表）、系统文件表和信息节点表，就很难理解I/O重定向。这些表都是Linux系统维护的数据结构。进程文件表中包含每个进程已打开文件指向系统文件表的指针，而系统文件表中的文件指针则指向位于内存中的信息节点表，信息节点表包含文件的信息节点，而信息节点含有文件的打开方式（如读或写等）、文件的当前读写位置，以及怎样获取文件的数据内容等信息。文件描述符是文件在系统文件表中的位置索引。每个进程的文件读写操作都是通过文件描述符实现的。操作系统利用文件描述符，从系统文件表和信息节点中找出目标文件，最终完成文件的处理，如图7-1所示。

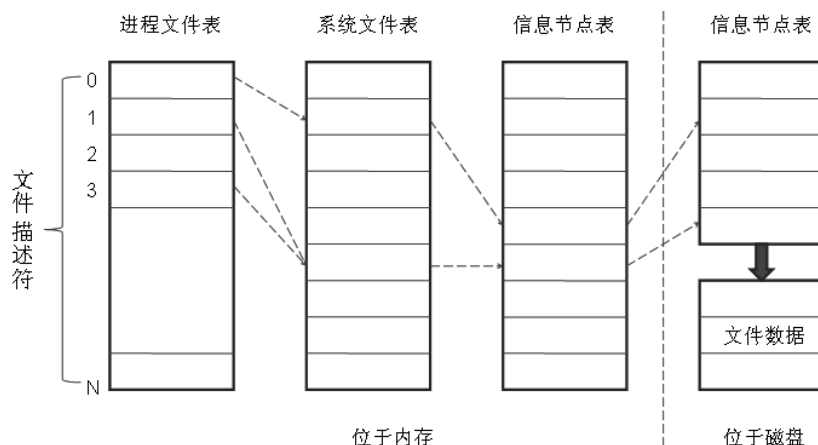


图7-1 文件描述符与文件的关系示意图

文件描述符0是标准输入，通常为终端的键盘输入。文件描述符1为标准输出，文件描述符2为标准错误输出，两者通常为终端显示。当注册到Linux系统时，这三个文件描述符是已经打开的默认文件描述符。

通常情况下，Shell通过标准输入读取命令语句，通过标准输出显示命令执行的结果。当执行过程中发现问题时，则通过标准错误输出显示错误信息。

在输入输出重定向的情况下，Shell需要执行若干处理动作：关闭指定的文件描述符，如关闭文件描述符0（如果标准输入被重定向）或关闭文件描述符1（如果标准输出被重定向）等；打开指定的文件；把新的文件指针放入文件指针数据结构表中刚才腾出的位置，以替代刚关闭的相应文件。

下面是一个I/O重定向的例子。

```
command < datafile
```

在这个例子中，标准输入被重定向到datafile中。在实现过程中，Shell首先关闭文件描述符

0，然后以输入方式打开datafile文件，最后把打开操作产生的文件指针信息复制到原标准输入所在文件指针数据结构表中已腾出的位置，如图7-2所示。

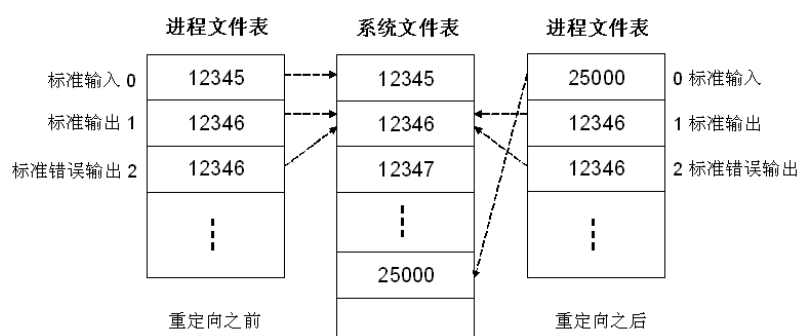


图7-2 标准输入重定向

至此，当命令从标准输入（文件描述符0）读取数据时，将会从新打开的datafile文件，而不是终端的键盘输入中读取数据。如果标准输出或标准错误输出也被重定向，Shell将采取同样的处理动作。

7.5.7 变量替换

变量替换涉及Shell内部变量、用户自定义变量、命令行参数或位置参数等变量替换。

变量替换表达式由美元符号“\$”与随后的变量名构成，如\$var。位置参数替换表达式由美元符号“\$”与随后的数字构成，如\$1、\$2、……等。变量名或参数名可以用花括号括起来，如\${var}。变量替换表达式前后也可以加单、双引号，如fname="\${var}"。但在实际替换时，两者存在细微的差别。详见7.2节有关变量替换的介绍。

7.5.8 算术运算结果替换

在Bash中，算术运算结果替换的语法格式如下：

```
$((expression))
```

当命令行中出现“\$((expression))”形式的基本元素时，Shell首先需要计算双括号中的表达式，然后使用计算结果替换“\$((expression))”。例如：

```
$ echo There are $((60*60*24*365)) seconds in a year
There are 31536000 seconds in a year
$
```

7.5.9 命令替换

命令替换表达式是由反向单引号“`”或\$(...)形式的命令语句组成的，命令的输出就是命令替换表达式的值。命令可以是任何Linux命令，包括普通命令、顺序执行的命令、组合命令或含有管道符号的并列命令等。

作为命令替换表达式的一部分，命令语句可以使用任何Shell机制，如变量替换和文件名生成机制等。因此，在执行命令之前，Shell需要把整个命令行完全展开，最终形成实际上能够执行的具体命令。

如果在命令的替换过程中出现多余的空格、制表符或换行符等，Shell将会在第二次重读、命令语句解析，以及执行引用字符处理时予以删除。因此，如果这些空格、制表符或换行符等应当保留，输入时应使用双引号括住整个命令替换表达式。

在反向单引号之前增加转义符号“\”，可以实现命令替换表达式的嵌套。下面的例子解释了命令表达式的嵌套是怎样实现的：

```
$ cat is.today
for fname in ${*}
do
    appoint=`grep \`date +%m/%d/%y\` ${fname}`
    echo "${appoint}"
done
$ cat calendar
11/08/09          Finish unit 15
11/09/09          Finish unit 16
11/10/09          Finish unit 17
$ date +%m/%d/%y
11/08/09
$ is.today calendar
11/08/09          Finish unit 15
$
```

在上述is.today脚本文件中，最内层的反向单引号各增加了一个转义符号“\”。这个脚本首先确定当天的日期，然后利用这个日期作为模式，使用grep命令检索fname变量指定的文件。最后把检索到的内容赋予变量appoint，并输出最终结果。

7.5.10 单词解析

在完成了变量替换、命令替换以及算术运算结果替换之后，Shell将利用IFS变量设定的字符作为分隔符，对经过各种替换后生成的命令行再次进行解析，以分离出最基本的命令行元素或单词。此时，如果命令行中存在单引号或双引号，其中的内容（包括null参数）将被看做一个单词。否则，Shell将会删除额外的空格、制表符和换行符，以及隐含的null参数。

IFS变量的默认值为空格、制表符和换行符，如果需要，用户也可以修改或重新定义IFS变量设置。

在IFS单词解析处理阶段，变量赋值语句是受特殊保护的。除了赋值之外，Shell不会对变量赋值语句做文件名生成等其他任何解析处理。因此，如果希望把一个变量表达式的值赋予等号左边的变量，而变量表达式又可能生成多个单词时，这样的变量表达式前后不必加双引号。

下面的例子解释了IFS变量的设置如何影响命令行的解释执行：

```
$ a=x:y:z
$ cat $a
cat: x:y:z: No such file or directory
$ IFS="$IFS:"
$ cat $a
cat: x: No such file or directory
cat: y: No such file or directory
cat: z: No such file or directory
$
```

7.5.11 文件名生成

命令行解释执行的下一步是扩展元字符，以生成文件名。在这个步骤里，Shell将会检索解析出来的每一个单词，检查其中是否包含任何符合文件名生成规则的元字符，如星号“*”、问号“?”和由方括号“[...]”表示的字符范围等。

对于包含元字符的每一个基本单词，Shell将会尝试匹配当前目录或指定目录中的文件名。如果找到匹配的任何文件名，Shell将使用这些文件名替换命令语句中的元字符表达式。例如：

```
$ ls file*
file1 file2 file3
$ echo file*
file1 file2 file3
$ rm file?
$ echo file*
file*
$
```

在文件名生成过程中，涉及到下一节将要介绍的双引号的引用处理。在下面的例子中，为了引用当前目录下的所有文件，我们首先为变量files赋值一个星号“*”：

```
$ files=*
$
```

当使用echo命令显示变量files的值，且在变量的前后增加了双引号时，Shell只做变量替换，但不执行文件名生成：

```
$ echo "${files}"
*
$
```

如果未加双引号，Shell将会扩展元字符“*”，执行文件名的生成过程。例如：

```
$ echo ${files}
file1 file2 file3
$
```

如7.6.10节所述，变量赋值时并不执行文件名生成。因此，在执行赋值语句“files=”时，变量files的值只是一个星号“*”，而不是当前目录下的所有文件名。在此步骤中，Shell不会解析双引号括住的变量赋值，但参数和变量替换会照常执行。如果是单引号，Shell将会禁止一切替换。因此，如果想用一个带有元字符的变量值，而又不希望扩展元字符，可用双引号括住变量表达式。

特别需要注意的是，文件名的生成是在变量替换、命令替换和/O重定向之后进行的。因此，在涉及文件名生成的表达式中要特别小心，尤其是在带有I/O重定向的表达式中，一定要注意。

7.5.12 引用字符处理

引用字符处理的目的是删除引用符号。在此处理步骤中，Shell开始删除转义符号“\”和单双引号等引用字符，展开引号中的表达式，完成命令执行前的所有预备工作。

引用字符处理涉及到文件名的生成及各种替换。如果一个基本单词前后有双引号，Shell将会禁止文件名生成，禁止除参数和变量替换之外的所有替换。如果有单引号，Shell将会禁止文件名生成和所有的替换。下面的例子解释了单引号、双引号以及不加任何引号的引用字符处理与文件名生成：

```
$ ls file*
file1 file2 file3
$ var=file*
$ set | grep var=
var='file*'
$ echo '$var'
$var
$ echo "$var"
file*
$ echo $var
file1 file2 file3
$
```

7.5.13 进程替换

进程替换是相对于命令替换而言的。命令替换的主要作用是把命令的输出数据赋予某个变量，以便做进一步的处理，或者利用Shell的管道机制，直接接收某个命令的输出数据，或者为其他命令直接提供输入数据等。例如，“var=`command`”或“command | command”。

进程替换则是利用/dev/fd/<nn>特殊文件或管道文件，把一个进程的输出结果，作为输入数据提交给另一个进程。换言之，进程替换是利用/dev/fd/<nn>特殊文件或管道文件，实现进程之间标准输入与标准输出的交互通信的。

进程替换采用“<(command)”或“>(command)”的形式实现。当圆括号中的进程开始运行时，其标准输入和标准输出将会连接到/dev/fd目录中的两个特殊文件或管道文件，同时把文件名作为参数传递给当前命令的标准输入或标准输出。如果采用的是“<(command)”形式的进程替换，当前命令的标准输出将会写到圆括号中指定进程的标准输入。如果采用的是“>(command)”形式的进程替换，作为参数传递的特殊文件或管道文件可用于读取圆括号中指定进程的标准输出。



注意 在“<”和“>”与圆括号之间没有空格，否则将会出现错误（信息）。

进程替换的一个重要功能特性是可用进程替换命令行中的文件名参数。当一个文件名参数位置出现“<(command)”形式的进程调用时，将会引起Shell执行圆括号中的命令，把进程的标准输出写到一个特殊文件或管道文件中，然后把此文件作为当前命令的标准输入，从指定进程的标准输出中读取输入数据。

同样，当一个文件名参数位置出现“>(command)”形式的进程调用时，也会引起Shell执行给定的命令，但Shell将会把该进程的标准输出写到一个特殊文件或管道文件中，然后以此文件作为圆括号中指定进程的标准输入。

因此，可以使用进程替换比较两个不同的命令，或者同一命令但选项或参数不同时的输出结果。例如，下列例子说明了怎样使用进程替换比较两个目录中的文件有何异同：

```
$ diff <(ls -l doc) <(ls -l doc2)
2c2
< -rw-r--r-- 1 qqxing qqxing 498 Nov  8 11:09 readme
---
> -rw-r--r-- 1 qqxing qqxing 516 Nov  8 11:10 readme
$
```

下列命令将会按照文件的大小，对“ls -l”命令输出的数据进行排序（其中，“-n”选项表示按数值排序，“-k 5”表示按输入数据的第5列排序）：

```
$ sort -n -k 5 <(ls -l)
total 128
-rw-r--r-- 1 qqxing qqxing 11742 Nov  8 11:53 handler.c
-rw-r--r-- 1 qqxing qqxing 11802 Nov  8 11:52 atmmmon.c
-rw-r--r-- 1 qqxing qqxing 18222 Nov  8 11:50 atmcom.c
-rw-r--r-- 1 qqxing qqxing 23221 Nov  8 12:05 atmstat.c
-rw-r--r-- 1 qqxing qqxing 55596 Nov  8 11:56 listener.c
$
```

sort命令的“-o”选项本来用于指定一个输出文件，以便存储排序后的结果。下面的例子以进程替换的形式代替输出文件，借用gawk命令，从排序后的结果中抽取文件的名称与大小字段：

```
$ sort -n -k 5 <(ls -l) -o >(gawk '{print $9 "\t" $5}')
handler.c          11742
atmmmon.c          11802
atmcom.c           18222
atmstat.c          23221
listener.c         55596
$
```

7.5.14 环境处理

命令行解释的第14步是环境处理，其中包括变量赋值，检索PATH变量指定的目录，找出命令文件的存储位置等。

在完成变量赋值之后，Shell将根据PATH环境变量的定义，从左到右依次检索命令文件的存储位置，然后以命令文件的完整路径名替换命令行中的命令名。如果命令名中（最后一个字符位置之前）包含斜线字符“/”，Shell将会绕过PATH变量检索，假定给定的命令是一个规范的路径文件名——相对路径文件名或绝对路径文件名。

7.5.15 执行命令

至此，终于到了真正开始执行命令的时候了。如果命令行要求执行的是一个普通的Linux命令，Shell将启动一个单独的子进程执行相应的命令。如果命令行要求执行的是一个Shell内部命令，由Shell直接执行。

如果命令是一个经过编译后生成的应用程序，Shell启动的子进程将使用exec系统调用执行应用程序。如果是一个Shell脚本，Shell将解释执行脚本文件中的每一行语句。具体执行过程是，如果访问权限表示命令文件是可执行的，为了确定命令文件是否能够直接运行，子进程将会尝

试把命令文件加载到内存中，然后开始执行。如果无法加载到内存，子进程将假定相应的命令文件是一个Shell脚本，因而由Shell采用解释执行的方式运行脚本文件。

如果程序是采用前台方式运行的，Shell将会一直等待，直至子进程执行结束。如果程序采用的是后台运行方式，Shell将不再关心子进程的执行是否终止，而是立即在标准输出上显示一个作业号和进程ID，开始继续处理其他请求。

7.5.16 跟踪执行过程

如果先前使用set命令的“-v”或“-x”等选项设定了跟踪命令的执行过程，Shell将会在这个处理步骤中输出当前正在执行的命令语句，同时在每个命令语句之前插入一个加号“+”前缀，然后再输出命令的运行结果。

因为命令替换已在命令执行之前完成，此时输出的是实际执行的命令，而且是按照命令执行的逻辑顺序依次输出的。

如果命令行包含管道形式的并列命令，并列命令的执行顺序并没有严格的定义，命令的调度执行将是随机的。通常，Shell会首先调度执行管道字符之前的第一个命令，然后依次调度执行其他后续命令，但这并不等于每个命令都是严格按照其在命令行中出现的位置顺序执行的。

第8章 Shell高级编程

除了顺序执行命令之外，Shell还提供丰富的控制结构语句，其中，if-then-else和case等转移控制语句使用户能够通过条件测试与逻辑运算等机制，控制Shell脚本的执行流向，利用for、while和until等循环语句能够实现各种循环控制。

本章主要介绍Shell的各种控制结构语句，讨论怎样利用控制结构语句，编写功能完善的Shell脚本，介绍怎样利用Shell的编程机制和Linux系统提供的丰富命令，创建自己的日常系统维护脚本。在此基础上，建议读者仔细阅读利用Shell脚本实现的各种系统维护命令，编写出高水平的Shell脚本。

8.1 if条件语句

在Shell脚本中，if语句是最基本的，也是最常用的重要语句之一。利用if-then-else条件语句，可以实现脚本执行流程的控制转移。最简单的if语句语法格式如下：

```
if command
then
    command-list
fi
```

if语句以给定命令的出口状态作为判断条件，确定转入哪一个控制流向。如果命令执行成功，返回一个0值的出口状态，则执行紧随then之后的命令。如果命令执行失败，返回一个非零值的出口状态，则执行紧随fi之后的命令。fi表示if语句的结束。

if-then之间的命令可以是一个普通命令，也可以是一组命令；可以是一个简单的测试语句，也可以是一组复杂的组合测试语句。当if-then之间存在一组命令时，if-then控制结构测试的是if和then之间最后一条命令的出口状态。

8.1.1 if语句的基本形式

许多if语句均使用test命令作为逻辑判断条件，根据test命令的测试结果控制Shell脚本的流向，或决定下一步应采取的动作。例如，许多Shell脚本都会在正式开始运行之前检查命令行的参数个数是否正确，如果给定的参数数量不足，则显示Shell脚本的用法，然后结束运行。基于这一思路，如果Shell脚本要求最少提供两个参数，可以在Shell脚本的开始部分写出下列if语句（其中的“\$#”表示命令行参数的个数）：

```
$ cat chkargs
if test $# -lt 2
then
    echo "Usage: chkargs arg1 arg2 ..."
    exit 1
fi
echo "Go on to the next."
$
```

“iftest ...”结构等价于“if [...]”和“if [[...]]”结构。因此，上述if语句可以改写如下：

```
if [ $# -lt 2 ]
then
    echo "Usage: chkargs arg1 arg2 ..."
    exit 1
fi
```

if语句也可以使用else子句处理test语句返回非零出口状态的情况，使之执行else子句后面的语句。完整的if语句语法格式如下：

```
if command
then
    command-list
else
    command-list
fi
```

例如，当运行某个进程（假定为atmmon）时，如果要把进程的运行结果记录到系统日志文件中，可根据进程的出口状态是否为零确定其运行是否正常结束，然后利用logger命令把结果写入系统日志文件。据此，可以写出下列代码：

```
atmmon                                # 自己开发的ATM监控程序
if [ test $? -eq 0 ]
then
    logger "atmmon has run successfully."
else
    logger "atmmon terminated abnormally."
fi
```

if语句后面既可以没有test语句，也可以没有“[...]”或“[[...]]”等测试结构，而是利用其他命令的执行结果作为测试条件。按照定义，这种形式也是符合if语句的语法规定的。

假定仅当某个目录存在，系统才会支持某一功能，因而才能执行其中的命令时，可以首先使用cd命令尝试进入指定的目录。如果cd命令执行成功，Shell脚本才能正常运行。否则，退出Shell脚本。根据这个思路，可以写出下列Shell脚本：

```
dir=/opt/atm
if cd "$dir" 2>/dev/null
then
    echo "Now we are in $dir."
    .....
else
    echo "Can't change to $dir."
    exit 1
fi
```

在“if command”结构语句中，给定命令的出口状态用于控制Shell脚本的转移流向。下面的例子说明了怎样利用diff命令返回的出口状态判断文件比较的结果。

```
if diff fname1 fname2 > /dev/null 2>&1
then
    echo "Files fname1 and fname2 are identical."
```

```

else
    echo "Files fname1 and fname2 differ."
fi

```

下面是一个比较有用的“if-grep”结构。其中，grep命令的“-q”选项用于抑制命令的输出。

```

if grep -q Linux fname
then
    echo "File contains at least one occurrence of Linux."
fi

```

当if、then以及条件测试命令位于同一行上时，then之前必须加一个分号。尽管if和then是一个整体，if和then是其中的两个关键字，但它们均表示一个子句的开始。加分号的目的是表示if子句的终止。这同多个命令语句位于同一命令行时中间必须加分号“;”分隔符是完全一样的，例如：

```

if [ condition ]; then
    command-list
else
    command-list
fi

```

8.1.2 嵌套的if语句

if语句及其条件测试结构还可以嵌套。第一种嵌套结构的if语句的语法格式如下：

```

if [ condition1 ]
then
    if [ condition2 ]
    then
        command-list
    fi
fi

```

上述情况相当于使用&&逻辑运算符组合条件测试表达式：

```

if [ condition1 ] && [ condition2 ]
then
    command-list
fi

```

第二种嵌套结构的if语句采用elif（else if的缩写）子句实现复杂的条件转移，其语法格式如下：

```

if [ condition1 ]
then
    command-list
elif [ condition2 ]
then
    command-list
else
    default-command
fi

```

通过使用elif子句, if语句可以实现多重测试结构。同样, 如果elif子句后面的测试条件的出口状态为真, 则执行紧随elif子句中then后面的语句。

例如, 若想在用户注册时, 根据系统的当前时间显示不同的问候语, 可以在profile初始化文件中增加下列语句:

```
$ cat greeting
now=`date +%H`
if [ ${now} -lt 12 ]
then
    greeting="Good morning."
elif [ ${now} -lt 18 ]
then
    greeting="Good afternoon."
else
    greeting="Good evening."
fi
echo "$greeting"
exit 0
$ greeting
Good morning.
$ date
2009年 12月 22日 星期二 10:18:26 CST
$
```

每个嵌套的if语句必须以fi结束。作为一个组合语句, 整个if语句的出口状态是then或else子句中执行的最后一条命令的出口状态。如果没有执行任何命令, 其出口状态为零。

在下面的例子中, 由于给定的文件不存在(文件名有误), ls命令将会返回一个非零的出口状态。因此, Shell脚本不会执行最后一个echo语句。

```
$ cat ifthen
fname=/etc/password          # 正确的文件名应为/etc/passwd
LANG=C                       # 采用英文语言环境, 确保提示信息的准确性, 下同
if echo "This statement always return true."
    ls -l $fname
then
    echo "Come here."
fi
$ ifthen
This statement always return true.
ls: cannot access /etc/password: No such file or directory
$
```

现在让我们交换if与then之间的两个语句。尽管ls命令将会返回一个非零的出口状态, 但由于echo命令总是能够成功地执行, 故Shell脚本将会执行到最后一个echo语句, 然后才结束Shell脚本的执行。

```
$ cat ifthen2
fname=/etc/inittab           # Ubuntu Linux系统不用这个文件
LANG=C
if ls -l $fname
```

```

        echo "This statement always return true."
    then
        echo "Come here."
    fi
$ ifthen2
ls: cannot access /etc/inittab: No such file or directory
This statement always return true.
Come here.
$

```

8.1.3 if语句综合应用实例

下面是一个用于增加用户的shell脚本，其目的是避免直接使用useradd系统命令，在命令行中输入较多的选项与参数。利用此脚本，调用时只需提供一个用户名，如“add-user username”，即可增加指定的用户与用户组。其中采用了if-then、if-then-else和if-grep等结构语句，这里列出来供读者参考：

```

$ cat add-user
#!/bin/bash
if [ $# -ne 1 ]
then
    echo "Usage: add-user username"
    exit 1
fi
username=$1
comment=$1
homedir=/home/$username
minuid=1000

grep "^$username" /etc/passwd >/dev/null 2>&1
if [ $? -ne 0 ]
then
    alluids=`cat /etc/passwd | cut -d: -f3 | sort -n | tr '\n' ' '`
    set $alluids
    while [ $1 -le $minuid ]
    do
        shift
    done
    uid=$minid
    sum=$#
    while [ $sum -ge 1 ]
    do
        uid=`expr $uid + 1`
        if [ $uid -eq $2 ]
        then
            shift
            sum=$((sum-1))
            continue
        fi
        if ! grep -q $uid /etc/group
        then

```



```

                                break
                        fi
                done
        else
                echo "add-user: the username $1 exist."
                exit 2
        fi
        echo "Adding user: uid = $uid, gid = $uid, username = $username"
        useradd -u $uid -d $homedir -m -c $comment -s /bin/bash $username
        exit 0
$

```

8.2 case分支语句

case分支语句能够提供多路分支转移控制功能。case语句利用一个变量作为测试条件，变量可以具有多个值，不同的数值能够引起不同的程序走向。case语句的语法格式如下：

```

case "$variable" in
    pattern1)
        command-list ;;
    pattern2)
        command-list ;;
    .....
    patternN)
        command-list ;;
esac

```

Shell使用给定变量（variable）的值与每一个模式（pattern）依次进行比较。一旦发现匹配的模式，Shell将会立即执行紧随模式之后的所有命令，直至遇到双分号“;”结束。在双分号前的最后一个命令执行结束之后，Shell将跳转至紧随esac语句之后的第一个语句，开始继续执行。如果用做测试条件的变量值与任何模式都不匹配，则直接跳转到紧随esac语句之后的第一个语句处开始继续执行，而不执行case语句中的任何命令。

每个模式之后必须加一个右括号“)”，以便与随后的一组相关命令分隔。case语句中的每一组命令或代码块之后必须以双分号结束（最后一组命令或代码块除外），这相当于告诉Shell已执行到相关命令或代码块的边界。

模式中可以使用的元字符，也可以使用运算符“|”，表示多个模式的逻辑或关系。由于元字符“*”可以匹配任何数值、字符或字符串，因此可以用做默认的匹配模式。这个“万能的模式”必须作为case语句的最后一个模式，因为一旦检查到匹配的模式，Shell将会停止执行case语句的模式匹配检查，致使其他模式永远得不到匹配的机会。

下面是一些与case语句语法格式有关的说明：

- 因为case语句中用做测试条件的变量值通常都是一个单独的数值或单词，不会出现包含分隔符的字符串，故测试变量前后可以不加双引号；
- 每个用于匹配检查的模式之后必须附加一个右圆括号后缀；
- 除了最后一组命令或代码块，其他模式的相关命令或代码块之后均需附加一个双分号；
- 整个case语句以esac（case的反序拼写）结束。

Shell中的case控制结构相当于C/C++中的switch语句，允许整个控制结构根据条件测试变量的值执行相应的一组命令或代码块。因此，case控制结构是一种非常适合建立多路分支转移程序流向的工具，也能够容易地创建选择菜单。实际上，case控制结构是if-then-else嵌套结构的一种简化形式。

例如，利用case控制结构，我们可以设计一个简单的网络环境设置脚本netchoice，针对家庭ADSL网络环境、办公室网络环境以及Samba测试环境，使用不同的网络接口配置文件替换/etc/network/interfaces配置文件，然后重新运行新的/etc/init.d/networking启动脚本。运行netchoice脚本时，既可以采用菜单选择的形式，又可以采用命令行参数的形式，适当地设置网络环境（由于修改interfaces文件，运行networking启动脚本等需要具有超级用户的访问权限，故需使用sudo命令，参见第9章“用户管理”），示例代码如下：

```
$ cat netchoice
#!/bin/bash
echo -e "\n\t----- Network Setup Menu ----- \n"
1.  Home/ADSL Networking Environment
2.  Office Networking Environment
3.  NFS/Samba Testing Environment
4.  Stop Home/ADSL Networking\n
Please enter your choice: \c"

read -n 1 choice
case "$choice" in
    1 | h )    echo -e "\nStarting ADSL ....."
               cp /etc/network/if.home /etc/network/interfaces
               /etc/init.d/networking restart
               pon dsl-provider
               ;;
    2 | o )    echo -e "\nSetting up DHCP ....."
               cp /etc/network/if.office /etc/network/interfaces
               /etc/init.d/networking restart
               ;;
    3 | n )    echo -e "\nSetting up NFS/Samba ....."
               cp /etc/network/if.nfs /etc/network/interfaces
               /etc/init.d/networking restart
               ;;
    4 | s )    echo -e "\nTerminating ADSL ....."
               poff
               ;;
    * )        echo -e "\nInvaild choice."
esac
exit 0
$ sudo netchoice

----- Network Setup Menu -----

1.  Home/ADSL Networking Environment
2.  Office Networking Environment
3.  NFS/Samba Testing Environment
4.  Stop Home/ADSL Networking

Please enter your choice:
```

在Linux系统的各种日常管理与Shell脚本维护中,按照使用的频繁程度,case语句也许是一种仅次于if-then-else语句的控制结构。尤其是/etc/init.d或/etc/rcN.d目录中的各种系统服务启动脚本,case语句是最常见的固定控制结构:利用同一个Shell脚本,在系统的启动、关机或重启过程中,分别以start、stop或restart等作为参数,启动、关闭或重启各种系统服务的守护进程。

下面是取自Apache服务器启动脚本的部分内容,其中就采用了典型的case控制结构用法:

```
$ cat /etc/init.d/apache2
.....
case $1 in
    start)
        log_daemon_msg "Starting web server" "apache2"
        if $APACHE2CTL start; then
            if check_htcacheclean ; then
                log_progress_msg htcacheclean
                start_htcacheclean || log_end_msg 1
            fi
            log_end_msg 0
        else
            log_end_msg 1
        fi
        ;;
    stop)
        if check_htcacheclean ; then
            log_daemon_msg "Stopping web server" "htcacheclean"
            stop_htcacheclean
            log_progress_msg "apache2"
        else
            log_daemon_msg "Stopping web server" "apache2"
        fi
        if apache_wait_stop; then
            log_end_msg 0
        else
            log_end_msg 1
        fi
        ;;
    .....)
esac
.....
$
```

在case命令语句中,还可以使用命令替换作为测试变量,以命令的输出作为测试变量的值。例如,Ubuntu 8.xx Linux系统提供的mountall.sh脚本曾使用“uname -s”命令的输出作为case语句中的测试变量值,以便在相应版本的系统中做特定的设置和处理。示例如下:

```
$ cat /etc/init.d/mountall.sh
.....
case "$(uname -s)" in
    *FreeBSD)                                # 当“uname -s”命令输出的后面为
        INITCTL=/etc/.initctl              # FreeBSD时,执行左边的赋值语句
        ;;
    *)
        INITCTL=/dev/initctl
```

```

        ;;
    esac
    .....
$

```

8.3 for循环语句

for语句是Shell中的一种基本循环控制结构。针对for语句中的每个参数，可以重复执行一组命令或代码块。for语句的语法格式如下：

```

for var [ in word-list ]
do
    command-list
done

```

其中，word-list是一系列参数，或称参数表，中间以空格为分隔符。对于参数表中的每一个参数值，重复执行一次do与done之间的命令。do与done之间的循环体command-list是一个代码块，可以是一个命令、一组命令，也可以是各种控制结构语句，包括for循环结构语句。

利用for循环结构，只要控制条件为真，即可重复执行一组命令或代码块。在每次循环过程中，Shell将会从给定的参数表中，依次取出一个参数值，并把参数值赋予给定的变量var，然后重复执行for循环体中的代码块。

下面以一个简单的例子说明for循环结构的执行过程：

```

for var in arg1 arg2 ..... argN
do
    echo $var
done

```

在上述例子中，第一次循环时把arg1赋予var变量，相当于执行“var=arg1”变量赋值语句，然后运行“echo \$var”命令；第二次循环时执行“var=arg2”变量赋值语句，然后再次运行“echo \$var”命令；依次类推，第N次循环时执行“var=argN”变量赋值语句，然后运行“echo \$var”命令。经过N次循环之后，整个for循环语句执行结束。

下面的例子进一步说明了for循环结构的执行过程。当提交一个命令，为了弄清执行的命令究竟位于哪个目录，尤其当存在不同版本的同名命令，而这些命令又处于不同的命令检索路径时，可以使用PATH变量中设置的命令检索路径，找出第一个发现的命令位于哪一个目录，即可得到答案。PATH变量中包含一系列由冒号分隔的目录，Shell正是利用PATH变量，检索用户输入的命令的：

```

$ cat where
IFS="{IFS}:"
flag=0
for dirname in `echo ${PATH}`
do
    if test -x "${dirname}/${1}"
    then
        echo "${dirname}/${1}"
        flag=1
    fi

```

```
done
if [ $flag -eq 0 ]
then
    echo "The $1 does not exist."
fi
$
```

在执行上述Shell脚本时，其输出结果如下：

```
$ where echo
/bin/echo
$ where findf
The findf does not exist.
$
```



如果do与for位于同一行上，则do之前应加一个分号分隔符。

```
for var in [word-list] ; do
```

下面的for循环利用diff命令，对当前目录下的C文件alpha.c、beta.c和gamma.c与/home/gqxing/src目录中的同名文件进行比较

```
for fname in alpha beta gamma ; do
    diff ${fname}.c /home/gqxing/src/${fname}.c
done
```

参数表中的参数也可以包含元字符，故也可以利用Shell的“*”、“?”或“[...]”等通配符建立参数表。例如，我们可以利用元字符重写先前的for循环语句，比较当前目录与/home/gqxing/src目录中的每一个C文件：

```
for fname in *.c
do
    diff $fname /home/gqxing/src/$fname
done
```

参数表中的每个元素还可以包含多个参数。当需要按组处理参数时，这种形式是非常有用的。在此情况下，可以使用set命令，强制解析参数表中的每一个元素，把其中的每个参数分配到一系列位置参数中。

例如，在下面的for循环语句中，参数表中的每个元素本身包含两个参数：

```
for planet in "Mercury 36" "Venus 67" "Earth 93" "Mars 142" "Jupiter 483"
do
    set $planet                                # 解析变量planet, 设置位置参数
    echo "$1 is $2,000,000 miles away from the sun."
done
```

此外，也可以使用命令替换生成for循环中的参数表。例如，假定因故需要修改当前目录及其子目录中所有文件的属主与同组属性，我们可以利用“ls -R”命令能够递归地列出各级目录中所有文件的特点，生成文件名参数，然后再使用chown和chgrp命令修改文件的属性。例如：

```
for fname in `ls -R`
do
```

```

        if [ -f $fname ]
        then
            chown gqxing "${fname}"
            chgrp gqxing "${fname}"
        fi
    done

```

下面是另外一个使用命令替换生成for循环语句参数表的例子。这个Shell脚本的功能是从指定（或当前）目录中检索文件的大小超过1 MB的文件（注意，由于find命令“-size”选项的参数值后面未加任何后缀，故默认的数据单位是512字节的数据块，而ls命令“-s”选项的数据单位是1 KB）。

```

$ cat bigfile
#!/bin/bash
dir=${1-`pwd`}
echo "Big file(in 1KB-block) under directory \"$dir\""
cd ${dir}
for i in `find . -size +2048 -print`
do
    if [ -f "${i}" ]
    then
        ls -s "${i}"
    fi
done
exit 0
$ bigfile /boot
Big file(in 1KB-blocks) under directory "/boot"
7476 ./initrd.img-2.6.31-14-generic
1628 ./System.map-2.6.31-14-generic
1676 ./grub/unicode.pf2
3800 ./vmlinuz-2.6.31-14-generic
$

```

在for循环语句中，“in word-list”部分可以省略，在此情况下，for循环语句需要使用运行Shell脚本时提供的命令行参数作为参数表。这种for循环结构尤其适用于编写一种通用的Shell脚本，针对给定的任何参数，执行同样一组处理命令。其简化的语法格式如下：

```

for variable
do
    command-list
done

```

假定公司有一个员工的电话号码簿，其数据内容如下：

```

$ cat phonebook
Cathy      617-495-1585      Boston
Jim        650-723-2300      Stanford
Ruth       516-367-8800      New York
Tom        410-516-8000      Baltimore
.....
$

```

我们可以写出下列Shell脚本，用于查阅任何给定员工的信息：

```
$ cat findname.sh
#!/bin/bash
for name
do
    if grep $name phonebook
    then
        :
    else
        echo "$name is not in the phonebook."
    fi
done
$
```

例如，为了查阅Cathy、Jim和XXX的电话号码，我们可以按照下列方式运行Shell脚本：

```
$ findname.sh Cathy Jim XXX
Cathy      617-495-1585      Boston
Jim        650-723-2300      Stanford
XXX is not in the phonebook.
$
```

整个for循环语句的输出可以重定向或通过管道输出到一个文件、命令或一组命令。根据这个特点，我们可以利用管道和sort排序工具，对上述的bigfile脚本做少许改进，把整个for循环的输出作为sort命令的输入，使得Shell脚本能够按文件的大小从大到小排序，然后顺序列出指定目录（或当前目录）中大于1MB的文件。示例代码如下：

```
$ cat bigfile2
#!/bin/bash
dir=${1:-`pwd`}          # 使用指定的或当前目录
cd $dir
echo "Big files(in 1KB-block) under directory \"$dir\""

find . -size +2048 2>/dev/null |          # 找出容量大于1MB的文件
while read file
do
    if [ -f "${file}" ]
    then
        ls -s "${file}"
    fi
done | sort -rn           # 按文件容量从大到小排序
exit 0

$ bigfile2 /boot
Big files(in 1KB-block) under directory "/boot"
7476 ./initrd.img-2.6.31-14-generic
3800 ./vmlinuz-2.6.31-14-generic
1676 ./grub/unicode.pf2
1628 ./System.map-2.6.31-14-generic
$
```

8.4 while循环语句

while语句的功能是根据一定的条件，循环执行一组特定的命令。while循环结构在循环体的前部测试执行条件。只要控制条件的测试结果为真（返回值为0），便继续执行循环体中的命令，否则立即结束整个while循环。与for循环相比，while循环更适合循环次数事先不确定的情况。其语法格式如下：

```
while [ condition-is-true ]
do
    command-list
done
```

下面是一个简单的while循环结构，其目的是求出1~100之间所有整数的总和。当\$Number变量的值大于100时，Shell将会跳到done后面的echo语句开始执行，输出\$sum变量的值，也即1~100之间所有整数的总和，其代码如下：

```
$ cat sum100
number=1
sum=0
max=100

while [ "$number" -le "$max" ]
do
    sum=`expr $sum + $number`
    number=`expr $number + 1`
done
echo $sum
$
```

同for循环一样，如果do与while位于同一行上，do与测试条件之间也应加一个分号分隔符：

```
while [ condition-is-true ] ; do
```

在while循环结构中，测试条件可以是一个简单的test语句，也可以是由多个条件表达式与逻辑运算符组成的组合测试语句；可以是一个普通的Linux命令，也可以是一组命令。循环体中可以包含任何命令语句，当然也包括任何结构语句。

在while循环结构中，当控制条件由多个命令语句组成时，实际上只有最后一个命令语句的出口状态才能决定是否继续执行循环体。也就是说，while语句仅仅检查do之前最后一条命令的返回值，如果返回值为0，则继续执行do与done之间的代码体；如果返回值大于0，Shell将会终止执行整个while循环语句，然后立即跳转并执行紧随done之后的命令语句。

在下面的例子中，用做while循环判断条件的第一个语句，即echo命令总是能够正常地运行。而read命令的特性恰好可以用做while循环结构的判断条件，当读取一个Ctrl-D组合键时，read命令将会返回一个非零的出口状态，因而可用于结束while循环：

```
$ cat lookup
while
    echo "\nEnter name to searched (Press Ctrl-D to end): \c"
    read name
```



```

do
    if test "${name}" = ""
    then
        continue
    else
        lookup.sql "${name}" # 这里的lookup.sql是一个MySQL脚本，用于检索数据库
    fi
done
exit 0
$

```

如果在关键字done后面增加一个重定向符号“<”，while循环结构的标准输入可以重定向到一个文件。另外，while循环结构的标准输入也可以由一个管道提供。

下面是一个利用管道为while循环结构提供数据，由while循环体逐行读取cat命令输出的数据，并对数据进行加工处理的例子（这里的处理仅为显示读取的数据）：

```

$ cat check
#!/bin/bash
line_num=1
cat $1 |
while read text
do
    echo "${line_num}: ${text}"
    line_num=`expr ${line_num} + 1`
done
exit 0
$

```

作为测试数据的文件内容如下（选自泰戈尔《Stray Birds》）：

```

$ cat stray.birds
"What language is thine, O Sea?"
"The language of eternal question."
"What language is thy, Osky?"
"The language of eternal silence."
$

```

运行上述脚本后，输出结果如下：

```

$ check stray.birds
1: "What language is thine, O Sea?"
2: "The language of eternal question."
3: "What language is thy, Osky?"
4: "The language of eternal silence."
$

```

8.5 until循环语句

until语句的功能是在一定的条件下，循环执行一组特定的命令。until循环结构也是在循环体的前部测试循环控制条件的。但与while循环结构不同的是，until循环结构测试的是终止条件。如果控制条件的测试结果为真（返回值为0），则立即结束整个until循环。如果控制条件的测

试结果为假（返回值大于零），便继续执行循环体中的命令，直至控制条件的测试结果为真，这一点恰与while循环相反。until语句的语法格式如下：

```
until [ condition-is-true ]
do
    command-list
done
```

同for和while循环一样，如果do与条件测试语句位于同一行上，则中间需要加分号分隔符。

```
until [ condition-is-true ] ; do
```

下面是一个采用Euclid算法，利用until循环求取最大公约数的例子。首先选取第一个参数作为被除数并赋予dividend变量，第二个参数作为除数赋予divisor变量。在until结构语句的每一次循环过程中，再把当前的除数赋予dividend变量，余数赋予divisor变量，作为下一次运算的因子。重复上述运算，直至余数为零，最后一个dividend即为最大公约数。整个循环运算过程结束。以下是该例子的代码：

```
$ cat gcd
#!/bin/bash
if [ $# -ne 2 ]
then
    echo "Usage: `basename $0` Number#1 Number#2"
    exit 1
fi
dividend=$1
divisor=$2
remainder=1
until [ "$remainder" -eq 0 ]
do
    let "remainder = $dividend % $divisor"
    dividend=$divisor
    divisor=$remainder
done
echo "The greatest common divisor of $1 and $2 = $dividend"
exit 0
$ gcd 7856 9980
The greatest common divisor of 7856 and 9980 = 4
$
```

8.6 select循环语句

select循环结构源于Korn Shell，主要用于建立选择菜单。select菜单的最大特点是，编程人员只需提供作为菜单选择项的参数表，菜单的组织与表现形式则由select语句负责实现。select语句的语法格式如下：

```
select variable [in list]
do
    command-list
break
done
```

执行select语句时，Shell将提示用户根据参数表中的选择项，通过输入相应的数字做出选择。通常，select语句使用PS3环境变量的值（#?）作为命令提示符。如果需要，编程人员可在select语句之前，赋予PS3环境变量一个新的值，以定制命令提示符。

下面便是一个使用select语句建立菜单的例子。

```
$ cat menu
#!/bin/bash
PS3='Choose your favorite book: ' # 设置select语句使用的命令提示符
echo
select book in "The Da Vinci Code" "Angels & Demons" "The Godfather" "Rage of Angles"
do
    echo
    echo "Your favorite book is <$book>."
    break # 使用break语句退出select循环
done
exit 0
$
```

运行上述Shell脚本时，其处理结果如下：

```
$ menu
1) The Da Vinci Code          3) The Godfather
2) Angels & Demons            4) Rage of Angles
Choose your favorite book: 1

Your favorite book is <The Da Vinci Code>.
```

在select语句中，“in list”部分可以省略，如果省略了参数表，select语句需要使用运行Shell脚本时提供的命令行参数作为参数表。这种处理方式使编写的Shell脚本更通用，更灵活。

```
$ cat menu2
#!/bin/bash
PS3='Choose your favorite book: '
echo
select book
do
    echo
    echo "Your favorite book is <$book>."
    break
done
exit 0
$ menu2 "The Da Vinci Code" "Angels & Demons" "The Godfather" "Rage of Angles"
1) The Da Vinci Code          3) The Godfather
2) Angels & Demons            4) Rage of Angles
Choose your favorite book: 2

Your favorite book is <Angels & Demons>.
```

下面是一个在函数中利用select语句实现菜单的例子。其效果与用法同第一个例子完全一样。

```
$ cat menu3
#!/bin/bash
PS3='Choose your favorite book: '
choice()
{
    echo
    select book
    do
        echo
        echo "Your favorite book is <$book>."
        break
    done
}
choice "The Da Vinci Code" "Angels & Demons" "The Godfather" "Rage of Angles"
exit 0
$
```

8.7 嵌套的循环

嵌套的循环意味着循环体中包含内嵌的循环语句。外层循环体每执行一次循环，内层循环体就要执行一个完整的循环，直至外层循环结束。当然，也可以利用循环体中的break语句中断相应循环体的继续处理。

例如，下面的例子就是利用嵌套的循环，列出指定目录（或当前目录）的目录层次结构树。

```
$ cat tree.sh
#!/bin/bash
# Written by Rick Boivie. Revised and simplified by Jordi Sanfeliu (patched
# by Ian Kjos).
# Last revised by GQ Xing.
search () {
    for dir in `echo *`          # `echo *`用于列出当前目录下的所有文件
    do
        if [ -d "$dir" ] ; then    # 如果$dir是一个目录
            dirlevel=0            # 临时变量，用于记录目录层次
            while [ $dirlevel != $1 ] # 记录内部嵌套的循环
            do
                echo -e "| \c"      # 表示目录层次，中间不换行
                dirlevel=`expr $dirlevel + 1`
            done
            if [ -L "$dir" ] ; then    # 如果目录是一个符号链接
                echo "|--$dir" `ls -l $dir | sed 's/^.*'$dir' //'`
                # tt=`ls -l $dir`
                # echo "|--$tt | cut -f9-11 -d' '"
            else
                # 显示横向连接符号并列出原目录及其指向的目录名
                echo "|--$dir"        # 显示横向连接符号和目录名
                numdirs=`expr $numdirs + 1` # 增加目录计数
                if [ -x "$dir" ] ; then    # 检查目录执行权限
                    cd "$dir"
                    search `expr $1 + 1`    # 递归地调用自己
                    cd ..
                fi
            fi
        fi
    done
}
```

```

        else
            echo "Changed to above directory is denied"
        fi
    fi
done
}

cd ${1:-`pwd`}
echo "Initial directory = `pwd`"
numdirs=0
search 0
echo "Total directories = $numdirs"
exit 0
$ tree /var
Initial directory = /var
|--autofs
|   |--misc
|--backups
|--cache
|   |--apache2
|   |   |--mod_disk_cache
|   |--apt
|   |   |--archives
|   |   |--partial
.....
$
```

8.8 循环控制与辅助编程命令

这一节主要介绍影响循环行为的控制命令与辅助编程命令语句。

8.8.1 break和continue命令

break和continue循环控制命令基本上等同于C或其他编程语言中break和continue语句的功能。break命令用于跳出相应的循环体，终止循环体的继续执行。而continue命令则用于越过循环体中尚未执行的命令语句，结束本次循环，直接跳转到下一次循环迭代。

break命令的主要用途是立即停止执行当前的循环体，然后跳转到循环体外（即关键字done后面）的命令语句处开始继续执行。break命令的语法格式如下：

```
break [n]
```

break命令后面可以附加一个数字参数。一个简单的break命令只是终止执行最内层的循环体。但是，如果break命令后面有一个数字n，则“break n”命令能够跳出n层循环体，将控制转移到外部第n个关键字done后面的命令语句处开始继续执行。

continue命令的主要功能是越过本次循环中余下的所有命令语句，从循环体的第一条命令语句开始继续执行循环体。其语法格式如下：

```
continue [n]
```

类似于break, continue命令也可以附带一个数字参数。一个简单的continue命令只是终止执行当前循环中尚未执行的命令语句, 继续执行下一次循环。如果continue命令后面有一个数字n, 则“continue n”命令能够终止执行当前循环, 从嵌套的外部第n层循环体的起始命令语句开始继续执行下一次循环。



注意 break和continue命令只能用于for、while和until等循环语句的循环体中, 位于关键字do与done之间。

下面的Shell脚本说明了break和continue命令在while循环中的作用。其中的break和continue命令用于控制用户的输入过程。这个Shell脚本的功能是提示用户连续地输入由空格分隔的字符串数据, 并把接收的数据存储到一个文件中。直至用户输入quit时, 退出while循环。接着对文件中的数据进行排序, 最后把已排序的数据再存回原文件中。

```
$ cat whiledo
#!/bin/bash
> datafile
while :
do
    echo -e "Please enter data (enter 'quit' to end): \c"
    read response
    case "$response" in
        quit)      break ;;          # 停止输入, 结束Shell脚本的运行
        '')        continue ;;       # 继续读取下一个数据
        *)         echo "$response" >> datafile
    esac
done
sort -o datafile datafile           # 对文件中的数据进行排序, 排序后仍保存到原文件
exit 0
$ whiledo
Please enter data (enter 'quit' to end): 888888
Please enter data (enter 'quit' to end): 222222
Please enter data (enter 'quit' to end): 999999
Please enter data (enter 'quit' to end): 666666
Please enter data (enter 'quit' to end): quit
$ cat datafile
222222
666666
888888
999999
$
```

下面是一个使用break和continue命令直接跳出多层循环, 或从外层循环体继续执行下一次循环的例子:

```
$ cat levels
#!/bin/bash
while :
do
    echo "Entering level 1"
```

```

while :
do
    echo -e "\tEntering level 2"
    while :
    do
        echo -e "\t\tEntering level 3"
        echo -e "\t\t\tInput c to continue \c"
        echo -e "or b to break: \c"
        read cmd
        echo -e "\t\t\tHow many levels? \c"
        read level
        case ${cmd} in
            [bB]*) break ${level} ;;
            [cC]*) continue ${level} ;;
        esac
        echo -e "\t\tLeaving level 3"
    done
    echo -e "\tLeaving level 2"
done
echo "Leaving level 1"
done
exit 0
$ levels
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: C
            How many levels? 2
        Entering level 2
            Entering level 3
                Input c to continue or b to break: B
                How many levels? 2
    Leaving level 1
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: c
            How many levels? 3
Entering level 1
    Entering level 2
        Entering level 3
            Input c to continue or b to break: b
            How many levels? 3
$

```



“continue n”命令的组织、应用和理解都比较困难，应尽量避免使用。

8.8.2 true命令

true命令除了用于返回一个表示测试成功的返回值0之外，不执行其他任何动作。因此，true命令经常用于无限循环的循环结构。

事实上，true与冒号“:”命令可以相互替换。但“:”命令是一个Shell内置命令，其执行速度要快于Linux系统提供的外部命令true。

8.8.3 sleep命令

sleep命令使Shell能够暂时休眠一定的时间，然后再执行下一个命令。因此，为了在执行两个命令之间暂停一定的时间，然后再执行，可以在Shell脚本中使用sleep命令。sleep命令的语法格式如下：

```
sleep [n]
```

其中，选用的参数n是暂停继续执行的时间间隔（以秒为计量单位）。sleep命令经常用于保持屏幕输出内容的暂时稳定，例如，为了检查当前正在复制的文件大小的变化情况，或查看某个进程是否继续活动，以便得出某种判断时，可以使用下列命令：

```
$ while true
> do
> ls -l fname
> sleep 5
> done
.....
$
```

或

```
$ while :
> do
> ps -ef | grep proc_name
> sleep 5
> done
$
```

8.8.4 shift命令

shift命令用于修改位置参数的值。按照命令参数指定的数量，所有的位置参数逐次向左平移。最左边的位置参数逐一移走，最右边的位置参数逐一左移后清除。同时，位置参数的总数也相应地减少。\$0是脚本文件的名称，在左移的过程中不受影响。shift命令语句的语法格式如下：

```
shift [n]
```

其中，n是左移的数量，默认值为1。每执行一次“shift n”命令语句，整个位置参数都会左移n个位置，表示位置参数总数的 \$# 变量值相应地减少n，表示全部位置参数的 \$* 或 \$@ 变量也相应地删除最左边的n个参数。

下面的例子说明了shift命令的典型用法。其中，while语句总是对\$1变量进行测试。由于使用了shift语句，每次循环之后，\$1的变量值都会发生变化。case语句依次解析每个命令行参数，并输出相应的信息。这个脚本假定“-a”选项之后必须指定文件名参数。脚本利用while、case和shift语句连续地检测位置参数，直至匹配“-a something”或“-b”，然后执行相应的代码。


```

$ cat options
#!/bin/bash
while test "${1}" != ""
do
    case "${1}" in
        -a* )   fname=`echo ${1} | sed 's/-a//'\`
                echo "Option a selected"
                echo "File name is ${fname}"
                option_a="yes"
                ;;
        -b )   echo "Option b selected"
                option_b="yes"
                ;;
        esac
    shift
done
$ options -aindex
Option a selected
File name is index
$ options -b
Option b selected
$ options -aindex -b
Option a selected
File name is index
Option b selected
$

```

8.8.5 getopt命令

getopt命令主要用于解析命令行选项，检查选项的合法性，为Shell编程提供方便，其语法格式如下：

```
set -- `getopt optstring $*`
```

其中，optstring是一个包含所有合法命令选项的字符串。如果选项字母之后附有一个冒号“:”，则意味着相应的选项需要提供一个选项参数（选项字母与选项参数之间既可以存在空格分隔符，也可以直接连接在一起）。“—”特殊选项用于界定命令选项的结束。不管存在与否，Shell都会把所有的普通选项之后放置一个“—”选项。

下面是一个取自网上的应用实例，其主要功能是根据用户提供的参数，利用ftp实现文件的自动下载。其中说明了ftpget脚本是怎样利用getopt命令处理其合法选项“-h”、“-d”、“-c”、“-f”和“-m”，解析和验证命令行参数的。

```

$ cat ftpget
#!/bin/bash
CMDFILE=/tmp/ftp.$$
TMPFILE=/tmp/ftp2.$$
usage="Usage: $0 [-h rhost] [-d rdir] [-c ldir] [-f rfile:lfile] [-m fpattern]"
ftptops="-i -n -v"
set -f
set -- `getopt h:d:c:f:m: $*`
# echo $*

```

```

if [ "$#" -lt 2 -o "$?" -ne 0 ]; then
    echo $usage
    exit 1
fi
trap 'rm -f ${CMDFILE} ${TMPFILE}; exit' 0 1 2 3 15
echo "user gqxing 123456" > ${CMDFILE}
for i in $*
do
    case $i in
        -h) remhost=$2; shift 2;;
        -d) echo cd $2 >> ${CMDFILE};
            echo pwd >> ${CMDFILE};
            shift 2;;
        -c) echo lcd $2 >> ${CMDFILE}; shift 2;;
        -m) echo mget "$2" >> ${TMPFILE}; shift 2;;
        -f) f1=`echo "$2" | cut -d: -f1`; f2=`echo "$2" | cut -d: -f2`;
            echo get ${f1} ${f2} >> ${TMPFILE}; shift 2;;
        --) shift; break;;
    esac
done
if [ $# -ne 0 ]; then
    echo $usage
    exit 2
fi
echo quit >> ${TMPFILE}
cat $TMPFILE >> $CMDFILE
ftp ${ftptopts} ${remhost:-169.254.78.100} < ${CMDFILE}
cat ${CMDFILE} >> ftplog
rm -f ${CMDFILE} ${TMPFILE}
exit 0

$ ftpget -d docs -c /tmp -f design
Connected to 169.254.78.100.
220 (vsFTPD 2.2.0)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
250 Directory successfully changed.
257 "/home/gqxing/docs"
Local directory now /tmp
local: design remote: design
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for design (308522 bytes).
226 File send OK.
308522 bytes received in 0.01 secs (53802.0 kB/s)
221 Goodbye.
$

```

8.8.6 getopts命令

getopts命令是getopt命令的更新和替代品。最初推出getopts命令的本意是取代getopt命令，但实际上这两个命令一直共存于UNIX以及Linux系统中。getopts命令用于获取以减号“-”或加

号“+”为起始字符的命令选项、选项参数和命令参数。其语法格式如下：

```
getopts opts string [arg...]
```

在编写Shell脚本时，最好能够养成使用getopts或getopt命令获取命令行选项、选项参数和命令参数的习惯。利用getopts命令，用户可在运行Shell脚本时以任何顺序输入命令选项。不带选项参数的命令选项既可以单独给出，也可以组合给出，但本身带有选项参数的命令选项需单独给出。

getopts命令语句采用一个合法的选项字符串确定Shell脚本能够接受的命令选项。例如，下述getopts命令定义的合法选项字符串为ior，表示Shell脚本能够接受的命令选项为“-i”、“-o”和“-r”：

```
getopts ior string
```

每次调用getopts命令语句时，命令行中的选项就会依次存入给定的变量string中。因此，Shell脚本可以对变量string的值做适当的处理。

在运行Shell脚本时，最后一个命令选项与命令参数之间还可以插入一个位置参数“—”。这是一个标志，表示命令选项的结束和命令参数的开始。

如果在运行脚本时提供了非法参数，getopts命令将会输出下列形式的出错信息，同时把一个问号“?”字符赋予string变量：

```
script_name: illegal option -- option_letter
```

getopts命令的返回值取决于是否能够成功地获取命令选项。如果读取了命令选项（包括非法的选项），getopts将返回数值0；否则，当遇到第一个位置参数“—”，第一个非选项参数（即命令参数），或命令行结束标志时，getopts命令语句将返回一个非零数值。

下面的例子说明了怎样使用getopts命令语句处理简单的命令行选项：

```
$ cat getoptions
#!/bin/bash
LANG=C
while getopts abc var
do
    case ${var} in
        a) echo Option -a found ;;
        b) echo Option -b found ;;
        c) echo Option -c found ;;
    esac
done
exit 0
$ getoptions -b -a
Option -b found
Option -a found
$ getoptions -adc
Option -a found
getoptions: illegal option -- d
Option -c found
$
```

如果选项带有参数，则需要在相应的选项字母后面附加一个冒号“:”字符。在读取冒号

之后，getopts将会把选项后的选项参数字符串赋予OPTARG变量。注意，选项与选项参数之间可以有空格分隔符，也可以连接在一起。

如果一个选项要求提供选项参数而未提供，getopts语句将会输出下列形式的出错信息：

```
script_name: option requires an argument - options_letter
```

getopts语句执行结束时，OPTIND变量的值将会指向第一个非选项参数的位置参数，也即第一个命令参数。如果位置参数的值为“—”，则忽略之。为了便于脚本按惯例处理命令参数，应将\$1指向第一个命令参数，\$2指向第二个命令参数，如此，等等。因此，我们可以利用shift命令，使之左移适当的位置。按照上面的说明，我们只需左移“\$OPTIND-1”个位置，即可使\$1指向第一个命令参数。

在下面的例子中，Shell脚本可接受4个选项，其中“-o”和“-r”选项要求提供相应的参数。在运行Shell脚本时，任何一个选项既可以提供，也可以不提供。如果提供了“-o”或“-r”选项，则必须提供相应的参数。

```
$ cat getoptions2
#!/bin/bash
LANG=C
while getopts o:r:nt var
do
    case ${var} in
        o) output_file="${OPTARG}" ;;
        r) report_file="${OPTARG}" ;;
        n) number_option="yes" ;;
        t) title="no" ;;
        \?) exit 2 ;;
    esac
done
echo "Output file = ${output_file}"
echo "Report file = ${report_file}"
echo "Numbering option = ${number_option:-no}"
echo "Title option = ${title:-yes}"
echo "Arguments before shift: ${*}"
shift `expr ${OPTIND} - 1`
echo "Arguments after shift: ${*}"
exit 0

$ getoptions2 -tn -o ofile -r rfile unit.12
Output file = ofile
Report file = rfile
Numbering option = yes
Title option = no
Arguments before shift: -tn -o ofile -r rfile unit.12
Arguments after shift: unit.12

$ getoptions2 -tn *
Output file =
Report file =
Numbering option = yes
Title option = no
Arguments before shift: -tn unit.01 unit.02 unit.03
Arguments after shift: unit.01 unit.02 unit.03

$ getoptions2 unit.12
```

```
Output file =
Report file =
Numbering option = no
Title option = yes
Arguments before shift: unit.12
Arguments after shift: unit.12
$ getoptions2 -tn -o ofile -r
getoptions2: option requires an argument -- r
$
```

8.9 循环语句的I/O重定向

作为一个整体，while、until与for循环结构语句中的循环体（包括while或until与do之间的测试条件语句）也可以实现I/O重定向。对于循环结构语句来讲，所谓的I/O重定向主要是重定向循环体中的标准输入。当然，也可以利用I/O重定向和管道的方法，把循环体中的输出结果保存到文件中，以备将来查阅。

另外, 函数也可以采用同样的处理方式, 实现 I/O 重定向。为了实现标准输入的 I/O 重定向, 可在循环结构语句的结束标志 (如关键字 done) 后面附加重定向符号 “<” 和适当的输入文件。

8.9.1 while循环的I/O重定向

对于while循环结构语句而言，下列语法格式表示循环体中需要的所有输入数据均取自指定的文件。

```
while [ condition-is-true ]
do
    command-list
done < datafile
```

下面的例子是取自一个安装脚本的部分代码片段。由于while循环结构语句的标准输入已重定向到/etc/fstab文件，故循环体中read语句读取的数据均来自/etc/fstab文件。在while循环中，每执行一次read语句，即从/etc/fstab文件中读取一行数据，并把有效数据分别赋值到包括fsdev、mntpt、fstype、mntopts、fsdump和fspassno等在内的7个变量中。

这个脚本片段的目的是检查系统中是否存在独立的/var文件系统。如果存在/var文件系统,则利用fsck命令检测并修复/var文件系统。

```
# cat checkfs
#!/bin/bash
while read fsdev mntp fstype mntopts fsdump fspassno
do
#       if [ "${mntp}" = "/var" ]           # 如果发现/var文件系统，检测并安装
#       if [ "${mntp}" = "/mnt" ]           # 为了验证，这里改为/mnt
#       then
#               echo "The $mntp file system (${fsdev}) is being checked."
#               fsck -t ${fstype} -y ${fsdev}
#               break
#       fi
done < /etc/fstab           # 把read命令的标准输入重定向到/etc/fstab文件
exit 0                      # 为了验证，这个文件也做了修改
```

```
# checkfs
The /mnt file system (/dev/fd0) is being checked.
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
/dev/fd0: clean, 15/184 files, 759/1440 blocks
#
```

8.9.2 until循环的I/O重定向

until循环结构的I/O重定向与while循环结构几乎完全雷同，下面的脚本也是根据上述例子改写的。除了采用until循环结构，其功能完全一样。

```
# cat checkfs2
#!/bin/bash
# 直至遇到/etc/fstab文件结束标志才终止until循环
until ! read fsdev mnttp fstype mntopts fsdump fspassno
do
    if [ "${mnttp}" = "/"var" ]      # 如果发现/var文件系统，执行fsck命令，
    then                            # 然后退出until循环
        echo "The $mnttp file system (${fsdev}) is being checked."
        fsck -t ${fstype} -y ${fsdev}
        break
    fi
done < /etc/fstab                  # 把read命令的标准输入重定向到/etc/fstab的文件
exit 0
#
```

8.9.3 for循环的I/O重定向

for循环结构也可以实现I/O重定向，使循环体中的输入输出命令能够直接读取或写入指定的文件。许多日常维护任务就是利用I/O重定向，在不改写Shell脚本的情况下，只需修改输入文件，即可实现特定的系统维护目的。

在下面的例子中，我们的目的是找出指定目录或默认目录中两个月以来未曾访问过的文件。为了保存查询的结果，我们稍做变化，利用tee命令实现for循环体标准输出的I/O重定向，在显示检索结果的同时，把输出数据保存到/tmp/filelist文件中。示例代码如下：

```
$ cat oldfile
#!/bin/bash
dir=${1-`pwd`}                    # 使用指定的目录或当前目录
cd ${dir}
echo "Old files under directory \"${dir}\""
for file in `find . -type f -atime +60 -print 2>/dev/null`
do
    if [ -f "${file}" ]
    then
        ls -l "${file}"
    fi
done | tee /tmp/filelist          # 显示并保存查询结果
exit 0
$ oldfile /home/gqxing/docs
```

```

Old files under directory "/home/gqxing/docs"
-rw-r--r-- 1 gqxing gqxing 164248 2009-11-02 ./TechDesign
-rw-r--r-- 1 gqxing gqxing 192128 2009-11-02 ./UserGuide
-rw-r--r-- 1 gqxing gqxing 222699 2009-11-02 ./RefManual
-rw-r--r-- 1 gqxing gqxing 102666 2009-11-02 ./ReleaseNote
$

```

此外, if-then 条件转移语句中的代码块也可以实现 I/O 重定向, 只是实际意义并不大, 除非把 if-then 代码块放到一个循环结构中。

总之, 在 Shell 脚本中, 灵活地利用 I/O 重定向, 能够生成必要的运行报告和日志文件, 完整地记录脚本的处理过程, 对脚本的调试和以后的档案留存都有重要的意义。

8.10 Here 文档

here 文档是一种具有特殊用途的代码块, 是 I/O 重定向的一种特例。here 文档采用 I/O 重定向的方法, 把一系列需要从键盘上输入的命令, 模拟人工输入方式, 一行一行地提交给交互式应用程序或命令, 如 ftp 和 MySQL 数据库的 mysql 等。here 文档的语法格式如下:

```

program <<LimitString
command1
command2
.....
commandN
LimitString

```

其中, 特殊的 I/O 重定向符号 “<<” 与 “LimitString” 表示 here 文档的开始, 单独另起一行的第二个 “LimitString” 表示 here 文档的结束。 “LimitString” 是启动指定程序后中间一系列交互命令的分界符。I/O 重定向的效果是把 here 文档列出的命令提交给交互式应用程序或命令的标准输入。here 文档的作用相当于执行下列命令:

```

interactive-program < command-file

```

其中, command-file 包含一系列需要人工输入的命令或数据:

```

command1
command2
.....
commandN

```

下面是一个利用 here 文档维护数据库的例子。当 MySQL 数据库从一个服务器迁移到另一个服务器时, 在安装 MySQL 数据库软件之后, 首先需要赋予用户建立数据库的权限, 然后还需要把先前利用 mysqldump 命令备份的数据库、数据库表及其数据加载到新的 MySQL 数据库中。为了简化人工操作步骤, 可以使用下列 Shell 脚本实现数据库的重建:

```

$ cat makedb
#!/bin/bash

mysql -u root -psqladmin <<EOF
USE mysql;
GRANT ALL PRIVILEGES ON books.* TO gqxing@"localhost" IDENTIFIED BY 'a1b2c3';
FLUSH PRIVILEGES;

```

```
quit
EOF
mysql -u gqxing -palb2c3 books < /tmp/bookdump.sql
exit 0
$
```

在上述here文档中，GRANT语句表示赋予用户gqxing在当前系统中创建数据库的权限。here文档之后的mysql命令用于导入先前备份的数据库、数据库表及其数据。



选择字符串分界符“LimitString”时应确保其在Shell脚本中是唯一的，至少不应在其界定的一系列命令中间出现，以免产生混淆。

下面的例子说明怎样利用here文档运行vim编辑器，模拟vim编辑器的交互过程，输入“i”命令和ESC键，插入两行数据，最后把编辑的数据内容写入指定的文件。

```
$ cat emuvim
#!/bin/bash
if [ -z "$1" ]
then
    echo "Usage: `basename $0` filename"
    exit 1
fi

vim $1 <<EOF
i
I cannot choose the best.
The best choose me.
^[
ZZ
EOF
exit 0
$
```

使用下列命令运行Shell脚本时，即可得到一个自动编辑的文本文件：

```
$ emuvim fname
Vim: Warning: Input is not from a terminal
$ cat fname

I cannot choose the best.
The best choose me.

$
```



在上述Shell脚本中，“`^I`”是一个字符，表示Esc键。为了在vim编辑器中输入Esc键，可以先按Ctrl-V组合键，接着再按Esc键，即可得到一个单字符的“`^I`”字符。此外，上述编辑命令形式将会在实际数据内容前后各加一个空行。为了避免这种情况出现，可以对上述的编辑形式稍加修改，最终得到一个只含两行数据内容的文件。改造的结果如下：

```
vim $1 <<EOF
iI cannot choose the best.
The best choose me.^I
ZZ
EOF
```


对于非交互式的实用程序或命令，有效地利用here文档，有时也会取得非常好的效果。下面是一个利用here文档和cat命令输出多行信息的例子：

```
cat <<End-of-message
=====
The system ${NODENAME} will be shut down in ${time}.
Please logoff as soon as possible.
=====
End-of-message
```

事实上，如果单纯为了显示多行数据，大可不必使用here文档，只需使用简单的echo命令，即可实现多行信息的输出。here文档的主要用途在于模拟执行各种交互式的程序或命令。例如：

```
echo "
=====
The system ${NODENAME} will be shut down in ${time}.
Please logoff as soon as possible.
====="
```

here文档要求其中的输入数据，尤其是作为结束标志的字符串分界符“LimitString”必须位于单独另起一行的起始位置。为了使脚本的可读性更强，编程人员喜欢在数据行之前增加制表符，使得脚本错落有致。为了使here文档能够正确地读取数据，可在第一个字符串分界符之前增加减号“-”标志（如<<-LimitString），使之在读取数据时能够删除数据行前的制表符（但“-”标志并不影响空格，也不影响文本行中间的制表符）。例如：

```
$ cat heredoc
#!/bin/bash
cat <<-ENDOFMESSAGE
    I live in you, you live in me;
    We are two gardens haunted by each other.
    Sometimes I cannot find you there,
    There is only the swing creaking, that you have just left,
    Or your favourite book beside the sundial.
ENDOFMESSAGE
exit 0
$ heredoc
I live in you, you live in me;
We are two gardens haunted by each other.
Sometimes I cannot find you there,
There is only the swing creaking, that you have just left,
Or your favourite book beside the sundial.
$
```

here文档也支持变量和命令替换，因而能够把不同的参数传递到here文档的代码体中，相应地改变here文档的输出。下面是一个在here文档中使用变量替换的例子。

```
$ cat upload.sh
#!/bin/bash
if [ -z "$1" ]
then
    echo "Usage: `basename $0` Filename-to-upload"
```

```

        exit 1
    fi
    if [ ! -f "$1" ]
    then
        echo "Specified file is not exist."
        exit 2
    fi
    fname=`basename $1`          # 删除文件名的目录部分
    host="169.254.78.100"
    dir="/home/gqxing/docs"

    ftp -inv $host <<End-Of-Session    # “-n” 选项禁止执行自动注册过程
    user gqxing 123456
    put $fname $dir/$fname
    bye
    End-Of-Session

    exit 0
$ upload.sh design
Connected to 169.254.78.100.
220 (vsFTPD 2.2.0)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
local: design remote: /home/gqxing/docs/design
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
308522 bytes sent in 0.01 secs (57641.3 kB/s)
221 Goodbye.
$

```

为了禁止在here文档中使用变量替换或命令替换，可以在第一个字符串分界符前后增加单引号或双引号，或者在第一个字符串分隔符前增加转义字符。

下面就是一个禁止here文档执行变量替换的例子。

```

$ cat greeting2
#!/bin/bash
name="${1:-Zhang}"
from="the author of this script"
cat <<'Endofmessage'          # 使用 “cat <<'Endofmessage'”或使用cat <<\Endofmessage” ,
                                # 三者效果一样

Hello, $name.
Greetings to you, $name, from $from.
Endofmessage
exit 0
$ greeting2 wang

Hello, $name.
Greetings to you, $name, from $from.
$

```

禁止变量替换使here文档能够输出纯文字的文本，生成Shell脚本或其他程序代码（如C程序等）。例如，下面的代码用于生成一个新的Shell脚本。

```
$ cat gen.sh
#!/bin/bash
OUTFILE=generated.sh          # 生成的Shell脚本的文件名
(
    cat <<EOF
    #!/bin/bash
    echo "This is a generated Shell script."
    echo "and will be named: $OUTFILE"
    exit 0
    EOF
) > $OUTFILE

if [ -f "$OUTFILE" ]
then
    chmod 755 $OUTFILE        # 使生成的Shell脚本能够执行
else
    echo "Problem in creating file: \"$OUTFILE\""
fi
exit 0
$ gen.sh
$ cat generated.sh
#!/bin/bash
echo "This is a generated Shell script."
echo "and will be named: generated.sh"
exit 0
$
```

利用here文档和“.”命令，还可以注解掉Shell脚本中的代码，或增加自包含的文档数据。这种代码注解技术可用于调试Shell脚本。编程人员只需增加两行标志代码，即可临时注解掉大块的Shell代码，从而避免了在每行代码前增加一个注释符“#”，调试结束后再逐行地一一删除。其代码如下：

```
$ cat commentblock.sh
#!/bin/bash
: << COMMENTBLOCK
..... # Shell代码或文档
COMMENTBLOCK

: << DEBUGXXX
for file in *
do
    cat "$file"
done
DEBUGXXX
exit 0
$
```



here文档中的第二个字符串分界符必须占用单独一行，并位于行首，前后不能包括任何空白字符（包括空格和制表符等）。否则将会妨碍Shell的识别，导致无法预料的后果。

8.11 Shell函数

同其他编程语言一样，Bash等Shell也支持函数，尽管实现的功能有一定的限度。一个函数是一个子程序，这个程序中利用一组代码块，实现特定的处理功能。与Shell脚本相比，函数经常直接存储在内存中，因而执行速度要快于Shell脚本。

无论何处，只要存在一组需要重复执行的处理动作，或针对不同的参数，能够获取相应的返回值，都可以考虑使用函数。函数的语法格式如下：

```
function_name() {  
    command-list  
}
```

或

```
function function_name {  
    command-list  
}
```

第一种函数形式是C程序员比较熟悉的。如同C语言一样，函数的左括号也可以出现在第二行上。注意，如同下列函数定义形式所示，如果整个函数定义与其中的命令位于同一行上，左花括号之后与右花括号之前必须各存在一个空格。同时，每个命令之后除附加分号之外，至少保留一个空格。

```
function_name( ) { cmd1; cmd2; .....; cmdn }
```

在定义函数时，函数名不能与现有的变量同名。如果函数与变量同名，Shell将会给出一个错误信息。此外，函数名与左圆括号之间必须连在一起，中间不能有空格。这个圆括号相当于告诉Shell，这是一个函数定义。

例如，为了在一个Shell脚本中重复执行两个变量值的交换，我们可以定义下列函数：

```
exchange()          # 交换两个变量的值  
{  
    temp=${color1}  
    color1=${color2}  
    color2=${temp}  
    return  
}
```



在定义函数时，不能在函数体内使用exit命令，因为函数的执行与当前的Shell脚本同属一个进程。否则的话，当函数执行到exit命令时，将会终止整个Shell脚本的继续执行。

在函数定义中，表示整个函数执行结束的替代方法是利用Shell的内部命令return实现的。也就是说，函数中return语句的功能同Shell脚本中的exit命令一样，其本意是终止函数的执行。在Shell脚本中，只有在函数定义中才能使用return命令，但并非必需。实际上，仅当期望函数返回一个数值时，才真正需要使用return语句。如果函数中未用到return命令，则函数运行结束时的返回值就是函数体中执行的最后一条命令的出口状态。

同exit语句一样，return语句也可以带一个整数参数，以便在函数执行结束时返回给定的数值，同时将这个返回值赋予“\$?”变量。编程人员可以使用return语句显式地指定返回值。否则，函数的返回值就是函数中执行的最后一条命令的出口状态（如果命令执行成功则返回0，否则返回一个非0值的错误代码）。return语句的语法格式如下：

```
return [n]
```

在Shell脚本中，可以通过“\$?”变量引用函数的返回值。这一点与C语言等中的函数类似，但与C语言等不同的是，在调用Shell函数时，只需直接写出函数的名字，然后给出必要的参数，而无需使用圆括号。当遇到一个函数名时，Shell首先会检查给定的名字是否为一个函数，如果不是，再利用PATH环境变量按命令检索。

如同常规的函数调用或Shell脚本的运行方式一样，调用Shell函数时也可以提供参数。而函数可以处理传递给自己的参数，最终把处理结果返回Shell脚本，以便在脚本中做进一步的处理。例如：

```
function_name $arg1 $arg2 ..... $argN
```

下面是一个求取指定整数范围内所有素数的例子。对于任何给定的整数，我们首先排除偶数，仅对奇数进行处理。通过一个while循环，主程序把3、5，直至小于或等于给定整数的奇数依次传递给prime函数，prime函数则以给定的整数参数作为被除数，依次除以3、5，直至整数参数的平方根（取整）为止。如果中间出现余数为零的情况则返回0（说明给定的整数参数不是素数），否则返回1（说明给定的整数参数为素数）。

```
$ cat prime
#!/bin/bash
prime()
{
    declare -i num lim rem
    num=3
    lim=`echo $1 | awk '{print sqrt($1)}'`
    while [ $num -le $lim ]
    do
        rem=`expr $1 % $num`
        if [ $rem -eq 0 ]
        then
            return 0
        fi
        num=`expr $num + 2`
    done
    return 1
}
declare -i number
while true
do
    echo -e "\nEnter a number: \c"
    read number
    if [ $number -lt 2 ]
    then
        exit 1
    fi
done
```

```

fi
echo "The prime number list within $number"
echo -e "2 \c"
if [ $number -eq 2 ]
then
    echo
    exit 2
fi

i=3
while [ $i -le $number ]
do
    prime $i
    ret=$?
    if [ $ret -eq 1 ]
    then
        echo -e "$i \c"
    fi
    i=`expr $i + 2`
done
done
exit 0
$ prime

Enter a number: 200
The prime number list within 200
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163
167 173 179 181 191 193 197 199
Enter a number:
$

```

return语句的返回值仅限于0 ~ 255的整数范围内。为了返回较大的数值，可以使用echo等命令语句，然后采用命令替换的形式获取函数的返回值。例如，为了求取两个变量中的最大值，我们可以定义下列函数：

```

$ cat max.sh
#!/bin/bash
if [ $# -ne 2 -a $# -lt 2 ]
then
    echo "Usage: `basename $0` number#1 number#2"
    exit 1
fi
max() {
    if [ $1 -gt $2 ]
    then
        echo $1
    else
        echo $2
    fi
}
echo "The greatest number of $1 and $2 = `max $1 $2`"
$ max.sh 7856 9980

```

不管在Shell脚本中，还是在交互式Shell环境中，在调用函数之前，首先必须对函数进行定义。定义之后，函数将位于用户的Shell运行环境中。使用不带参数的set命令，可以看到函数的定义。使用unset命令，或终止当前运行的Shell，即可取消当前的函数定义，因而也就不能再执行了，示例代码如下：

函数也可以在Shell脚本中定义，在脚本中执行，最后随脚本的终止而取消。如果在Shell脚本中定义函数，函数定义可以出现在允许使用命令语句的任何位置。但是，函数的定义必须位于调用函数的语句之前。对于需要经常使用的函数代码，最好在用户自己的profile文件中定义，这将保证一旦注册，只要需要，随时都可以直接调用函数。

```
$ cat report.sh
#!/bin/bash
prompt()
{
    while
        echo "Please answer yes or no: \c"
        read yes_no
    do
        case "${yes_no}" in
            y* | Y*) yes_no="y"
                        break
                        ;;
            n* | N*) yes_no="n"
                        break
                        ;;
            *) echo
                continue
        esac
    done
}
```

```

;;
        esac
    done
}
echo "Would you like headings?"
prompt
if test "${yes_no}" = "y"
then
    echo "Headings selected."
    echo "Start to process ... done."
fi
echo "Would you like a summary only?"
prompt
if test "${yes_no}" = "y"
then
    echo "Summary only selected."
    echo "Start to process ... done."
fi
exit 0
$ report.sh
Would you like headings?
Please answer yes or no: y
Headings selected.
Start to process ... done.
Would you like a summary only?
Please answer yes or no: n
$

```

无论何时调用函数，都会重新设置当前Shell进程的位置参数，使调用函数时提供的参数成为新的位置参数。也就是说，Shell将会使用调用函数时提供的参数重新为位置参数赋值。如果调用函数时未提供参数，位置参数（中的值）将被清除。

函数按位置引用传递给自己的参数\$1、\$2、……。因此，也可以使用shift命令处理传递给函数的参数。有关位置参数与shift命令方面的内容，可以参考前面介绍的例子，这里不再细叙。

与其他编程语言不同，Shell函数通常只接受数值参数。如果把变量名作为参数传递给函数，函数将会把变量名作为字符串常量处理。也就是说，函数将按字符串文字常量解释变量参数，示例如下：

```

$ cat varref
#!/bin/bash
echo_var ()
{
    echo "$1"
}
Message=Hello
Hello="Good Morning"
echo_var Message
echo_var Hello
echo_var "${Message}"
echo_var "${Hello}"
exit 0

```



```
$ varref
message
Hello
Hello
Good Morning
$
```

事实上，函数也是一个代码块，这意味着也可以重定向函数的标准输入、标准输出和标准错误输出。例如，在下面的函数定义例子中，“<\$file”表示把函数的标准输入重定向到\$file变量指定的文件中。当调用function_name函数时，函数中的read语句不再读取来自键盘的输入数据，而是直接读取自\$file变量定义的文件：

```
#!/bin/bash
function_name()
{
    .....
    read lines
    .....
} < $file          # 重定向函数的标准输入
```

在下面的例子中，“function_name2 <\$file”语句表示仅在调用普通的function_name2函数时，才把标准输入重定向到\$file变量定义的文件中。因此，函数中的read语句读取的数据仍然取自\$file变量定义的文件，而非键盘输入：

```
#!/bin/bash
function_name2()
{
    .....
    read lines
    .....
}
function_name2 < $file
```

下面我们再通过一个冒泡排序的例子说明怎样调用函数，作为本节的结束。另外，这个例子涉及到一些数组方面的概念，读者可以参考8.13节的介绍。

冒泡排序算法的基本思想是：对于n个需要排序的元素，执行n-1次循环。每次循环均从第1个元素开始，依次比较相邻的两个元素。如果第i个元素大于第i+1个元素，则交换两个元素的位置。直至比较到最后第n-j+1个元素（其中j为循环次数）才结束。第一次循环结束后，最大的元素将会交换到最后一个数组元素位置。第2次循环结束后，仅次于最大元素的元素将会交换到最后第2个数组元素位置。依次类推，直至循环结束。在每次循环中，较大的元素依次后移，较小的元素依次前移。当最后一次循环结束时，整个数组元素将完成从小到大的排序。

```
$ cat bubble.sh
#!/bin/bash
exchange()          # 交换两个数组元素
{
    temp=${colors[$1]}
    colors[$1]=${colors[$2]}
    colors[$2]=$temp
    return
}
```

```

}
colors=(red green blue yellow black white brown rose grey)
echo "0: ${colors[*]}"          # 输出排序前的全部数组元素
elements=${#colors[@]}         # 求出数组元素的个数
let "count = $elements - 1"
pass=1
while [ "$count" -gt 0 ]
do
    index=0                    # 每次循环均从第一个数组元素开始
    while [ "$index" -lt "$count" ]
    do
        if [[ ${colors[$index]} > ${colors[`expr $index + 1`]} ]]
        then
            exchange $index `expr $index + 1`
        fi
        let "index += 1"
    done                      # 一次冒泡排序循环结束
    let "count -= 1"
    echo "$pass: ${colors[@]}"  # 显示每次循环后的中间排序结果
    let "pass += 1"
done                          # 整个冒泡排序循环结束
exit 0
$ bubble.sh
0: red green blue yellow black white brown rose grey
1: green blue red black white brown rose grey yellow
2: blue green black red brown rose grey white yellow
3: blue black green brown red grey rose white yellow
4: black blue brown green grey red rose white yellow
5: black blue brown green grey red rose white yellow
6: black blue brown green grey red rose white yellow
7: black blue brown green grey red rose white yellow
8: black blue brown green grey red rose white yellow
$

```

8.12 逻辑并列结构

命令的逻辑与（&&）和逻辑或（||）并列结构提供了一种依次执行一系列命令的手段，能够有效地替代复杂的、嵌套的if-then语句结构。

8.12.1 逻辑与命令并列结构

逻辑与命令并列结构的语法格式如下：

```
command-1 && command-2 && ..... && command-n
```

命令的逻辑与并列结构表示从第一个命令开始，依次执行每一个命令。如果当前命令的返回值（出口状态）为零，则继续执行下一个命令。如果中间某个命令的返回值大于零，则整个并列命令链的执行结束。也就是说，第一个返回非零值的命令即为逻辑与并列结构中最后执行的一个命令。

下面是取自/etc/init.d/autofs启动脚本中的一行代码。这一代码首先检查/etc/default/autofs文件是否存在，如果文件存在，则在当前的Shell中执行该文件，以便设置NFS自动安装守护进程

使用的环境变量:

```
[ -f /etc/default/autofs ] && . /etc/default/autofs
```

如果把逻辑与命令并列结构改为if-then结构语句, 上述命令相当于:

```
if [ -f /etc/default/autofs ]
then
    . /etc/default/autofs
fi
```

8.12.2 逻辑或命令并列结构

逻辑或命令并列结构的语法格式如下:

```
command-1 || command-2 || ..... || command-n
```

与命令的逻辑与并列结构相反, 逻辑或命令并列结构意味着从第一个命令开始, 依次执行每一个命令。如果当前命令的返回值(出口状态)大于零, 则继续执行下一个命令。如果中间某个命令的返回值为零, 则整个并列命令链的执行结束。也就是说, 第一个返回零的命令即为逻辑或并列结构中最后执行的一个命令。

下面是取自/etc/init.d/networking启动脚本中的一行代码。这一代码首先检查/sbin/ifup文件是否存在, 且是否可以执行, 如果文件存在且可以执行, 则从下一行开始执行启动脚本中的后续代码。如果文件不存在, 或即使存在但不能执行, 则结束启动脚本的继续执行, 这是因为系统需要使用ifup命令完成网络接口的配置。

```
[ -x /sbin/ifup ] || exit 0
```

如果把逻辑或命令并列结构改为if-then结构语句, 上述命令相当于:

```
if [ ! -x /sbin/ifup ]
then
    exit 0
fi
```



在逻辑与和逻辑或命令并列结构中, 整个结构语句的出口状态是其中最后执行的一条命令的出口状态。

灵活地使用逻辑与和逻辑或命令并列结构及其组合, 可以简化Shell编程, 但有时也会增加理解的困难, 因而需要做额外的调试。

8.13 Shell数组

Bash等Shell还支持一维数组, 数组的大小没有限制。数组采用整数索引, 第一个数组元素从零开始。为了引入一个数组, 可以采用下列语法格式, 在声明一个数组的同时, 对数组进行初始化:

```
[declare -a] array_name=( element1 element2 ..... elementN )
array_name[index]=value
```

采用第一种语法格式定义数组，能够在声明数组的同时，对数组进行初始化。如果在定义数组时再增加一个“declare -a”前缀，则由declare语句预先声明的数组能够加速数组的处理，提高Shell脚本的运行性能。

例如，为了定义一个表示周日的数组，可以使用下列语句：

```
$ weekday=( Sunday Monday Tuesday Wednesday Thursday Friday Saturday )
$
```

采用第二种语法格式，即数组元素赋值的形式，逐个定义每一个数组元素，也可以构建一个数组。例如，我们可以采用下列方式定义并初始化一个等价的weekday数组：

```
weekday[0]=Sunday
weekday[1]=Monday
.....
weekday[6]=Saturday
```

实际上，定义任何一个数组元素，也可以构建一个数组。但是，如果仅对部分数组元素进行赋值，只能构建一个部分初始化的数组：

```
$ array=( [0]="first " [1]="second " [3]="fourth" )
$
```

按照上述方式定义并初始化数组之后，其结果如下：

```
$ echo ${array[0]}
first
$ echo ${array[1]}
second
$ echo ${array[2]}
$
$ echo ${array[3]}
fourth
$
```

在具体应用过程中，可以像操作普通变量一样操作数组元素，只是引用形式稍微不同而已。而且，许多标准的字符串操作也适应于数组元素。同样，为了引用数组元素的内容，也可以采用\${array_name[n]}等形式实现变量替换。

例如，在定义了weekday数组之后，即可使用数组变量名及其下标引用任何一个数组元素：

```
$ echo ${weekday[0]}
Sunday
$ echo ${weekday[1]}
Monday
$
```

与\$@或\$*变量值表示所有的位置参数类似，\${array_name[@]}或\${array_name[*]}变量值表示数组的所有元素。示例如下：

```
$ echo ${weekday[@]}
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
$ echo ${weekday[*]}
Sunday Monday Tuesday Wednesday Thursday Friday Saturday
$
```

同样，与`$#`变量值表示位置参数的数量类似，`${#array_name[@]}`或`${#array_name[*]}`变量值表示数组`array_name`的长度，也即所有数组元素的数量。下列例子说明数组`weekday`具有7个数组元素：

```
$ echo ${#weekday[@]}
7
$ echo ${#weekday[*]}
7
$
```



`${#array_name}`变量值仅表示第一个数组元素`array_name[0]`的长度。示例如下：

```
$ echo ${#weekday}
6
$
```

为了求取其他数组元素的长度（字符个数），可以采用`${#array_name[index]}`的变量替换形式，指定具体的数组元素。示例如下：

```
$ echo ${#weekday[0]}
6
$ echo ${#weekday[6]}
8
$
```

对于数组而言，某些Shell命令具有特殊的意义。例如，使用`unset`命令能够删除任何数组元素，甚至整个数组。

```
$ unset weekday[1]           # 相当于执行 “weekday[1]=” 语句
$ echo ${weekday[1]}

$ unset weekday
$ echo ${weekday[@]}

$
```

下面的例子说明了怎样利用数组和一定的算法，计算任何一天是星期几：

```
$ cat whatday
#!/bin/bash
typeset -i year month day mm=1 sum week yy
typeset -i leap=0
months=( 0 31 28 31 30 31 30 31 31 30 31 30 31 )
weekday=( Sunday Monday Tuesday Wednesday Thursday Friday Saturday )

month=${1:-`date '+%m'`}
day=${2:-`date '+%d'`}
year=${3:-`date '+%Y'`}

let yy=year-1
let sum=yy*365+yy/400+yy/4-yy/100

if [ `expr $year % 4` -ne 0 ]      # 如果年份不能被4除尽，则不是闰年
then
```

```

        leap=0
    elif [ `expr $year % 100` -ne 0 -o `expr $year % 400` -eq 0 ]
    then                                     # 如果年份能够被4除尽，但不能被100除尽，则是闰年
        leap=1                             # 如果年份能够同时被4、100和400除尽，则是闰年
    fi                                     # 如果年份既能被除尽4，又能被100除尽，但不能被
                                           # 400除尽，则不是闰年

    while [ $mm -lt $month ]
    do
        sum=`expr $sum + ${months[$mm]}`
        mm=`expr $mm + 1`
    done
    sum=`expr $sum + $day + $leap`
    week=`expr $sum % 7`
    echo "$month $day, $year is ${weekday[$week]}"
    exit 0
$ whatday
11 26, 2009 is Thursday
$ whatday 12 22 2009
12 22, 2009 is Tuesday
$

```

灵活地组合使用数组初始化语句“array=(element1 element2 …… elementN)”与命令替换，能够把一个文本文件的内容加载到数组中。因此，定义并初始化数组的第二种方法是命令替换法。采用命令替换法初始化数组，可以构建一个具有多个元素的数组，数组的大小依据命令输出的多少而定。

在采用命令替换形式初始化数组时，Shell使用空格、制表符和换行符等空白字符作为数组元素的分隔符，把每个字符串分配到一个数组元素中，最终构建成一个字符串数组。

例如，假定dict文件包含下列两行数据，采用命令替换形式的数组定义语句“array=(`cat dict`)”可以声明并初始化一个拥有7个元素的数组：

```

$ cat dict
Fedora  Gentoo  OpenSUSE  Ubuntu
FreeBSD  OpenBSD  NetBSD
$ array=( `cat dict` )
$ echo ${array[@]}
Fedora  Gentoo  OpenSUSE  Ubuntu  FreeBSD  OpenBSD  NetBSD
$ echo ${#array[*]}
7
$

```

定义并初始化数组的第三种方法是采用Shell元字符与文件名生成方法。采用元字符与文件名生成形式定义数组，也可以构建并初始化一个具有多个文件名元素的数组，数组的大小依据文件的数量而定。同使用命令替换形式初始化数组类似，Shell采用空格作为数组元素的分隔符，把每个文件名分配到一个数组元素中，最终构建成一个文件名数组。

例如，为了对当前目录下的C程序进行操作，我们可以使用下列语句生成一个文件名数组：

```
$ filelist=(*.c)
$ echo ${filelist[@]}
atmcom.c atmon.c atmstat.c handler.c listener.c
$ echo ${#filelist[@]}
5
$
```

此外，还可以把普通变量看做数组的一个特例（一个特殊的数组），即只有一个元素的数组。因此，即使并未明显地把变量声明为数组，Bash也允许以数组或数组元素的处理方式处理普通变量。示例如下：

```
$ string=abcABC123ABCabc
$ echo ${string[@]}
abcABC123ABCabc
$ echo ${string[*]}
abcABC123ABCabc
$ echo ${string[0]}
abcABC123ABCabc
$ echo ${string[1]}          # 一个普通变量只能看做第一个数组元素（下标为0）
$ echo ${#string[@]}        # 数组中只有一个元素，也即字符串本身
1
$
```

数组变量具有自己的语法规则，可以利用赋值语句实现整个数组的复制。也可以在数组的原有基础上增加新的元素。例如，下列两个语句均可实现整个数组的复制（其中的第一个语句实为包含变量替换的数组赋值语句）：

```
$ array2=( "${array1[@]}" )
$ array2="${array1[@]}"
```

而且，还可以使用下列语句，把一个数组和一个新的元素复制到另一个数组中：

```
$ array=( "${array[@]}" "new element" )
```

Bash仅支持一维数组，但稍加变通即可仿真多维数组。限于篇幅，这里不再赘述。

8.14 信号的捕捉与处理

信号是一种能够影响进程运行状态的外部事件，是由用户终端（如按下Ctrl-C组合键）、操作系统内核或其他进程（如kill命令）发送给当前或指定进程的消息，用于通知进程某种事件已经发生。进程可根据预定的方案，采取一定的处理措施或终止进程的执行。如果事先没有捕捉信号，默认的处理动作是停止进程的执行。

在Shell脚本中，trap命令用于指定需要捕捉的信号以及应采取的相应处理动作。当接收到某个指定的信号时，Shell将会立即触发相应的处理过程。trap命令的语法格式简写如下：

```
trap ["command-list"] [signal ...]
```

或

```
trap ['command-list'] [signal ...]
```

其中，`command-list`是无论何时收到指定的信号时都会立即执行的命令或一组命令。一旦命令执行结束，程序的控制逻辑将会恢复到因收到信号而中断的位置开始继续运行，除非命令中包含`exit`语句。如果在收到指定的信号时应当结束脚本的运行，可以使用`exit`语句作为指定命令的一部分（如果存在，`exit`语句通常应为命令组中的最后一个命令）。在通常情况下，指定的命令并非单个命令，而是一组命令，故前后应加单或双引号，命令中间以分号隔开。

`signal`表示准备捕捉的信号。信号可以是一个标准的信号名，也可以是一个数字。针对Shell编程而言，比较重要的、能够捕捉的常用信号只有0、1、2、3、15和20等。如果捕捉的信号是0（EXIT），则仅当脚本运行结束时才能执行相应的处理动作。这种例行处理方式经常用于清除Shell脚本运行过程中创建的临时文件。另外还有三个特殊的信号DEBUG、ERR和RETURN，主要用于调试和控制Shell脚本的执行，详见下一节的讨论。表8-1给出了这些信号的简单说明（至于系统定义的其他信号，详见第10章“进程管理”）。

表8-1 Shell脚本能够捕捉的信号

信号	信号名	简单说明
0	EXIT	进程结束信号。在Shell脚本的情况下，只要执行 <code>exit</code> 语句，终止Shell脚本的执行时，都可产生此信号。在Shell脚本运行正常结束时，即使没有明显地执行 <code>exit</code> 语句，也可产生此信号
1	SIGHUP或HUP	挂断。终端通信连接断开，或控制进程终止时产生的信号。通常，一旦捕捉到此信号，Shell脚本将会立即停止运行，除非使用 <code>nohup</code> 命令
2	SIGINT或INT	中断。按下中断键（也即Ctrl-C组合键）时产生的信号
3	SIGQUIT或QUIT	退出。按下Quit键（也即按下Ctrl-\或Ctrl-Shift-组合键）时产生的信号
15	SIGTERM或TERM	终止信号。这是kill命令产生的默认终止信号
20	SIGTSTP或TSTP	键盘停止信号。在交互方式下，通过按Ctrl-Z组合键停止当前进程时产生的信号。主要用于作业控制
	DEBUG	在执行Shell脚本中的每个命令之前，包括for、case和select命令语句，以及Shell函数中的第一个命令，都要运行trap语句中指定的命令
	ERR	Shell脚本中的任何命令，一旦运行结束时返回非零值的出口状态，都要运行trap语句中指定的命令。但while、until或if等结构语句中的测试语句除外
	RETURN	每当调用一个Shell函数，或利用“.”或source命令执行一个Shell脚本结束之后，都要运行trap语句中指定的命令

在Linux系统中，每个信号都有一个名字和数字。在编写Shell脚本时，建议使用名字指定捕捉的信号，这将使Shell脚本更具有可读性，也具有可移植性。



在Shell脚本中，有4个信号不应捕捉：其中信号9和19是不能捕捉的，而信号17和30则不应捕捉，这些信号的定义详见第10章“进程管理”。

利用trap命令，程序员可以采取下列三种处理措施之一：

- 捕捉指定的信号，然后执行必要的命令或处理动作；
- 忽略收到的信号；
- 清除先前的信号捕捉设置。

如果trap语句中指定的信号为0或EXIT，则当执行exit命令语句，或Shell脚本的执行正常结束时，立即执行trap语句中指定的处理动作。例如，下面的例子说明了怎样在Shell脚本运行结束时捕捉EXIT信号，从而显示脚本执行过程中设定的变量内容：

```
$ cat test.sh
#!/bin/bash
trap 'echo -e "Variable Setting:\na = $a\nb = $b"' EXIT
echo "This line will print before the \"trap\" statement."
echo "even though the \"trap\" statement occurs first."
echo
a=""
b=68
exit 0
$ test.sh
This line will print before the "trap" statement.
even though the "trap" statement occurs first.

Variable Setting:
a =
b = 68
$
```



上述Shell脚本的“exit 0”语句的存在与否并无太大的区别，因为在Shell脚本执行到最后一条命令语句时都会立即结束。如果最后一条命令语句的出口状态为0，其效果等同于执行“exit 0”语句。Shell脚本终止运行前执行的最后一条命令语句的出口状态即为整个Shell脚本的出口状态。

如果trap语句中指定的捕捉信号为1或HUP，且Shell脚本在运行过程中断开终端连接时，Shell将会立即执行定义的处理动作。如果trap语句中指定的信号为2或INT，当用户在Shell脚本执行过程中按下Ctrl-C组合键时，Shell将会立即执行定义的处理动作。如果trap语句中指定的信号为3或QUIT，当用户在Shell脚本执行过程中按下Ctrl-\组合键时，Shell将会立即执行定义的处理动作。

在下列例子中，如果在Shell脚本的运行过程中收到信号1、2或3，Shell将会执行trap命令语句中指定的命令序列，显示“Interrupted routine”，删除临时文件tmp\$\$，终止脚本的执行，最终返回一个数值1作为Shell脚本的出口状态：

```
trap 'echo Interrupted routine; rm -f tmp$$; exit 1' 1 2 3
```

或

```
trap 'echo Interrupted routine; rm -f tmp$$; exit 1' HUP INT QUIT
```

有时，我们不希望任何人中途打断Shell脚本的运行。因此，为了防止任何人使用Ctrl-C组合键打断Shell脚本的正常运行，可以使用下列形式的trap语句，屏蔽中断信号：

```
trap '' 2
```

或

```
trap 'echo "Ctrl-C disabled."' 2
```

如果trap语句的命令部分中没有指定任何处理动作，或明显地指定一个null值，则表示忽略指定的信号：

```
trap '' signal
```

或

```
trap ':' signal
```

上述两个trap命令存在细微的差别：第一个trap语句表示完全忽略指定的信号；第二个trap语句则意味着在收到指定的信号时不做任何处理。这两者在同一Shell脚本中的差别并不明显，但在父进程与子进程的处理方面则大不相同。

在第一种情况下，如果父进程忽略收到的指定信号，则子进程也将忽略相应的信号。例如，下面的例子表示父进程将会忽略捕捉到的中断信号（Ctrl-C组合键），同时告诉子进程也忽略这一信号：

```
trap "" 2
```

在第二种情况下，如果父进程捕捉到指定的信号，除了自己只是执行“:”语句而不做其他任何处理动作之外，还将告诉子进程清除其相应信号的设置，包括继承自父进程的信号设置（在收到信号2时执行“:”语句），同时把子进程未做“忽略”处理的所有信号恢复为默认的处理动作。在下列例子中，当捕捉到中断信号（按Ctrl-C组合键）时，父进程仅执行一个简单的“:”语句，但不会改变子进程对相应信号的默认处理动作：

```
trap ":" 2
```

在使用trap语句时，如果仅指定了准备捕捉的信号而未指定相应的处理动作，Shell将会删除先前为相应信号定义的处理动作。这意味着从此时开始，如果捕捉到这些信号，Shell将会按默认的处理动作执行。在下面的trap语句例子中，Shell将会删除为信号1、2和3定义的处理动作。也就是说，如果收到信号1、2或3，Shell将会执行默认的处理动作——终止Shell脚本的执行。

```
trap 1 2 3
```

通常，trap语句应是Shell脚本中的第一个语句。但程序员也可根据具体的编程要求做适当的调整。不管trap语句位于何处，在捕捉到指定的信号之前，Shell只是把trap语句读入内存，实际上并不执行。只有在收到指定的信号时，才会开始执行trap语句中定义的命令。

例如，在一个数据库更新的Shell脚本中，数据的输入过程并不重要，即使中断脚本的执行也无关紧要。但在数据库更新时，通常需要屏蔽一切外来的干扰，防止用户的中断信号等打断脚本的执行，以免破坏数据的完整性。因此，程序员可在数据库更新的关键代码处增加trap语句，使之屏蔽任何可能的外部信号干扰。在关键代码执行结束之后，再使用不带命令参数的trap语句清除之前的信号捕捉设置。示例代码如下：

```
$ cat updatedb
#!/bin/bash
```

```

while true
do
    echo "Please enter name, phone number, and email address"
    echo "separated by blank character (or enter quit to end):"
    read name phone email
    if test "${name}" = "quit"
    then
        break
    fi
    trap "" 2 3
    # Critical code segment to update database
    update.sql $name $phone $email
    trap 2 3
done
$

```

Shell脚本中的trap语句通常会读取两次。在开始执行Shell脚本时，Shell将会读取并存储整个trap语句，其间如有变量替换或命令替换，也将随之进行替换处理。当收到指定的信号时，Shell会再次读取并执行trap语句。同样，如果trap语句中仍然存在变量替换或命令替换，还会再次进行替换处理。

假定trap语句中存在变量替换表达式，如果变量是在脚本执行期间赋值的，则此情况下可能会存在一个问题：当第一次读取trap语句，并执行变量替换时，Shell将会采用初始的null值。

为了防止此类问题的发生，可在trap语句中使用单引号界定其中的命令，以避免在第一次读取trap语句时执行变量替换，使之仅当捕捉到指定的信号，需要执行相应的命令时再进行变量替换。

因此，下列两种命令引用形式是不同的：前者执行两次变量替换或命令替换，而后者只执行一次替换。

```

trap ["command-list"] [signal ...]
trap ['command-list'] [signal ...]

```

8.15 其他Shell课题

8.15.1 子Shell

当用户在键盘上输入单个命令时，Shell将以交互方式解释用户提交的命令，然后通过创建一个新的进程，执行用户提交的命令。而在运行Shell脚本时，当前Shell将会创建一个新的Shell进程，以批处理的方式处理用户提交的Shell脚本，解释执行脚本文件中的一系列命令。事实上，每个运行的Shell脚本都是当前Shell的子进程。

Shell脚本本身也可以创建子进程，也即子Shell。这些子Shell采用并行处理的方式，能够同时执行多个处理任务。通常，脚本中使用的每个外部命令都会使Shell创建一个相应的子进程，但内部命令则不然。由于这个原因，内部命令的执行速度明显快于等价的外部命令。

如下所示，位于圆括号中的命令将以子Shell的形式运行：

```
( command1; command2; command3; ... )
```

子Shell中的变量仅在子Shell代码块中有效，在调用子Shell的Shell中是不可见的，也就是说，子Shell中的变量属于本地变量。参见下面的例子：

```
$ cat subshell.sh
#!/bin/bash
outer_var=Outer
(
    inner_var=Inner
    echo "From subShell, \"inner_var\" = $inner_var"
    echo "From subShell, \"outer\" = $outer_var"
)

if [ -z "$inner_var" ]
then
    echo "inner_var undefined in main code body of Shell"
else
    echo "inner_var defined in main code body of Shell"
fi
echo "From main code body of Shell, \"inner_var\" = $inner_var"
# $inner_var will show as uninitialized because
# variables defined in a subShell are "local variables".
exit 0
$ subshell.sh
From subShell, "inner_var" = Inner
From subShell, "outer" = Outer
inner_var undefined in main code body of Shell
From main code body of Shell, "inner_var" =
$
```

在子Shell中使用cd命令所做的目录变动也不影响父Shell。也就是说，在子Shell中改变目录时，仅在子Shell中有效。这一点要特别注意。参见下面的例子：

```
$ cat subshell2
#!/bin/bash
echo "Current directory = `pwd`"
(cd /etc; cat timezone)
echo "Current directory = `pwd`"
exit 0
$ subshell2
Current directory = /home/gqxing/script
Asia/Shanghai
Current directory = /home/gqxing/script
$
```

8.15.2 Shell脚本的调试

Shell不提供Shell脚本的调试程序，甚至也不提供任何有关的调试命令或结构语句。脚本中的语法错误或明显的拼写错误产生的错误信息在调试脚本时经常无助于调试一个有误的Shell脚本。

下面是一个包含语法错误的Shell脚本，执行时将会出现下列错误信息：

```
$ cat test2.sh
#!/bin/bash
```

```

LANG=C
a=40
if [$a -gt 30 ]
then
    echo $a
fi
exit 0
$ test2.sh
test2.sh: line 4: [40: command not found
$

```

前面曾经讲过，方括号与表达式之间必须至少保留一个空格。而在上述脚本中，“[”和“\$a”连在一起，故Shell把“[40”看做一个单词，也即看做一个“[40”命令，而这样的命令显然是不存在的。

下面是一个漏掉关键字的例子。执行Shell脚本后产生的错误信息如下：

```

$ cat test3.sh
#!/bin/bash
LANG=C
for a in 1 2 3
do
    echo "$a"
# done                                # 不知何故注解掉了关键字done
exit 0
$ test3.sh
test3.sh: 8: syntax error: end of file unexpected (expecting "done")
$ cat test4.sh
#!/bin/bash
LANG=C
for a in 1 2 3                        # 遗漏了关键字do
    echo "$a"
done
exit 0
$ test4.sh
test4.sh: line 4: syntax error near unexpected token 'echo'
test4.sh: line 4: '    echo "$a"'
$

```



在输出产生语法错误的行号时，Shell是把注解行也计算在内的。另外，出错信息指出的行号也并不一定就是出错语句所在行的位置，但这一位置却是Shell解释程序最终发现脚本有误的地方。在上述第一个例子中，Shell读到最后仍然没有发现应有的关键字done，故给出的是一个根本就不存在的行号。

当遇到下述情况之一时，Shell脚本将会终止运行。

- 控制结构语句（包括循环结构语句和条件转移语句）中出现语法错误；
- 接收到某种信号，包括外部信号（如按Ctrl-C组合键、终端关闭或线路断开等）和内部信号；
- Shell内置语句（read和test语句除外）执行失败；

- 特殊替换中出现语法错误（如引号不匹配或不配对）；
- 赋值语句出现故障（如使用readonly命令定义的只读变量赋值）。

但下列情况不会引起Shell脚本停止运行。

- 试图执行一个不存在的命令（如命令名字拼写错误，或命令文件没有执行许可等）；
- I/O重定向故障（如试图使用“>”重定向符号写入一个已存在的文件等），此时，除了I/O受到影响外，并不影响脚本的执行；
- 命令异常中止（即使一个程序产生了内存映像文件core，Shell仍会继续执行下一个命令）；
- 命令终止运行时返回非零值的出口状态（除非使用trap语句跟踪此种情况）。

根据上述说明及实际应用，Shell脚本的错误情况可以归结为如下类型：

（1）运行时出现“syntax error”信息，说明脚本中存在诸如语句不完整，语法格式有误，遗漏关键字，引号不匹配或不配对等语法错误；

（2）可以运行，但最终的处理结果并非预期，这主要是由于逻辑错误造成的；

（3）可以运行，处理结果也无问题，但存在严重的边界效应。

为了调试Shell脚本，可以采用下列方法及工具：

（1）在脚本的关键位置加入echo语句，跟踪和显示关键变量的值，借以判断脚本是否存在逻辑问题；

（2）在关键位置使用tee命令检查进程或数据流；

（3）设置命令执行过程的跟踪标志（增加“-n”、“-v”或“-x”选项）：

- “sh -n scriptname”命令用于检查Shell脚本的语法错误，实际上并不运行脚本。这一命令形式等价于在脚本中插入“set -n”或“set -o noexec”语句。注意，这种检查并不能找出所有的语法错误。
- “sh -v scriptname”命令的作用是在执行之前显示每一个命令，然后显示命令的运行结果。这一命令形式等价于在脚本中插入“set -v”或“set -o verbose”语句。
- “-n”和“-v”选项可以一起工作。例如，“sh -nv scriptname”命令意味着给出详细的语法检查信息。
- “sh -x scriptname”命令的作用是以简化的方式显示每一条命令语句的运行结果，等价于在Shell脚本中插入“set -x”或“set -o xtrace”语句。
- 在脚本中插入并执行“set -u”或“set -o nounset”语句，使Shell在遇到试图使用未声明的变量时给出错误信息。

（4）使用trap命令捕捉Shell脚本终止执行时的位置。在Shell脚本中，一旦执行到exit命令，将会生成一个EXIT信号（0），并终止Shell脚本的执行。因此，捕捉EXIT信号，输出运行结束时的变量值通常是一种很有用的方法。

（5）在trap命令中，DEBUG信号使Shell能够在执行脚本中的每个命令之前立即执行指定的处理动作。这种“trap 'commands' DEBUG”设置能够在调试Shell脚本时跟踪变量的赋值情况，从中找出Shell脚本中的逻辑错误。

（6）在trap命令中，ERR信号使Shell能够在执行脚本中的每个命令时，跟踪运行过程中出现异常的命令。这种“trap 'commands' ERR”设置有助于找出Shell脚本中出错的命令语句，从而找出Shell脚本出错的原因。

(7) 在trap命令中, RETURN信号使Shell能够在调用脚本中的Shell函数, 或利用“.”或者source命令执行其他Shell脚本之后, 立即执行指定的处理动作。这种“trap ‘commands’ RETURN”设置能够跟踪Shell函数或其他脚本的执行情况。

信号捕捉机制对调试Shell脚本是很有用的。下面的例子就是利用“trap ‘commands’ DEBUG”设置, 在执行每个赋值语句之前输出var变量的变化情况:

```
$ cat testing
#!/bin/bash
trap 'echo " Trace Variable --> \${var} = \"${var}\""' DEBUG
var=66
let "var = 88"
exit 0
$ testing
Trace Variable --> $var = ""
Trace Variable --> $var = "66"
Trace Variable --> $var = "88"
$
```

如果trap语句中跟踪的信号为ERR, 则Shell脚本中的任何命令无论何时返回非零值的出口状态, Shell都会执行trap语句定义的处理动作。下面例子中的trap语句表示, 只要某个命令返回非零值的出口状态, 立即显示Filename变量的值。

```
$ cat testing2
#!/bin/bash
LANG=C
trap 'echo "$LINENO: Filename = \"${Filename}\""' ERR
ls -l somefile
exit 0
$ testing2
ls: cannot access somefile: No such file or directory
4: VAR Filename = ""
$
```

下面的例子说明, 不管脚本是否正常终止, 只要执行了exit语句或Shell脚本执行结束, 将会立即执行trap语句中定义的处理动作:

```
$ cat testing3
#!/bin/bash
# EXIT is the signal name generated upon exit from a script.
trap 'echo Variable Setting List: min = $min max = $max' EXIT
echo "This line will print before the \"trap\" statement."
echo -e "even though the \"trap\" statement occurs first.\n"

min=$1
max=$2
if [ $min -gt $max ]
then
    exit 1
fi
$ testing3 6 8
This line will print before the "trap" statement.
```

```
even though the "trap" statement occurs first.
```

```
Variable Setting List: min = 6, max = 8
```

```
$
```

下面是另外一个例子，说明怎样使用“set -v”命令，使Shell能够显示其读入的每一条命令（包括注释行），以及执行命令后的输出结果。需要说明的是，在调试较大的Shell脚本时，这种方法并不合适——面对满屏幕的命令与输出信息，第一个感觉恐怕就是头痛，简直无从下手。

```
$ cat trace.sh
#!/bin/bash
LANG=C
set -v
# This is a comment
date
echo ${TZ}
set +v
date
exit 0
$ trace.sh
# This is a comment
date
Wed Nov  5 00:52:50 CST 2008
echo ${TZ}

set +v                                # 关闭“-v”Shell执行过程跟踪设置
Wed Nov  5 00:52:50 CST 2008
$
```

下面的例子说明怎样使用“set -x”命令显示Shell读入并执行的每一条命令。与“set -v”命令不同的是，Shell将会在输出其读入的每个命令语句之前增加一个标记，如加号“+”字符（除了输出PS4的变量值作为标记之外，Shell还会在之前插入一个与PS4变量值第一个字符相同的字符），以区别于命令的输出数据。同时，Shell还会在输出命令之前，执行必要的变量替换、命令替换及元字符文件名生成，然后才显示命令与替换后的实际参数。

```
$ cat trace2.sh
#!/bin/bash
LANG=C
set -x
# This is a comment
date
dir=`pwd`
echo $dir
set +x
date
echo $HOME
exit 0
$ trace2.sh
+ date
Wed Nov  5 00:56:50 CST 2008
++ pwd
+ dir=/home/gqxing/script
+ echo /home/gqxing/script
```



```

/home/gqxing/script
+ set +x                                     # 关闭Shell执行过程跟踪设置
Wed Nov  5 00:56:50 CST 2008
/home/gqxing
$

```

原则上, Shell脚本允许使用未声明的变量。但利用“set -u”命令, 可以禁止Shell使用事先未声明的变量。利用这一特性, 可以发现变量名的拼写错误。下面的例子说明怎样使用“set -v”和“set -u”命令显示Shell执行的每一条命令, 一旦遇到事先未曾声明的变量, 将会停止Shell脚本的运行, 以便找出是否存在变量名拼写有误的错误。示例如下:

```

$ cat trace3.sh
#!/bin/bash
LANG=C
set -v
set -u
name=first
cat ${nmae}.file                          # 变量名拼写错误
wc -l ${name}.file
set +n
set +v
exit 0
$ trace3.sh
set -u
name=first
cat ${nmae}.file
trace3.sh: line 6: nmae: unbound variable
$

```

8.15.3 系统性能考虑

1. PATH变量的组织

PATH变量设置的目录顺序直接决定了命令检索的速度。每当执行一个非Shell的内置命令时, 系统都需要按照PATH变量指定的目录顺序检索命令。如果PATH变量设置的目录顺序不恰当, 将会影响命令的检索速度。一种典型的错误设置是把当前目录放到PATH变量的标准目录组织顺序的前面。这种设置造成的后果是, 每次执行命令时, 系统都会首先检索当前目录。这便引出两个问题: 当前目录下的程序究竟能够执行多少次? /bin和/usr/bin等目录中的命令需要执行多少次? 因此, 在下列两个PATH变量的目录顺序设置方式中, 一般应取前者而不是后者:

```

PATH=/bin:/usr/bin:..
PATH=./bin:/usr/bin

```

把当前目录放到PATH变量的前面, 对命令检索的时间影响究竟有多大, 还取决于当前目录的大小。如果当前目录较大, 将直接影响Shell的运行性能。还有一个实际问题, 如果把当前目录放到PATH变量的前面, 则无法创建与/bin或/usr/bin等目录中的命令同名的文件。

2. 文件的访问方式

当需要访问同一目录中的大量文件时, 最好利用cd命令首先进入该目录。在解释较长的目录文件名时, 操作系统需要从路径名的第一个目录开始, 逐层检索各级子目录, 直至找到最终

的文件名，因而需要花费较多的时间。如果给定的是相对于当前目录的文件名，操作系统只需直接访问给定的文件即可。

例如，下列两个脚本都是处理/home/icbc/credit/report/month目录中的所有文件，但因采取的方法不同，其运行效率也不相同。

```
for fname in /home/icbc/credit/report/month/*
do
    cat $fname
done

cd /home/icbc/credit/report/month
for fname in *
do
    cat $fname
done
```

上述第一种处理方法要求操作系统解释较长的路径名；而第二种方法虽然增加了一个语句，但操作系统只需在当前目录下直接访问每一个文件。

3. 内部命令与外部命令

作为解释程序的一部分，Shell提供若干内置命令语句。内置命令语句的执行速度明显快于外部的系统命令，其原因有三种。

（1）执行内置命令语句不必创建新的进程。对于提高运行效率，这是一个非常重要的关键因素。因为在运行一个命令时，耗费时间最多的是创建新进程，其过程包括分配内存空间，填写进程控制表，从磁盘中把程序调入内存等。

（2）执行内置命令语句时不需要按照PATH变量检索指定的目录。例如，当试图执行一个test命令时，Shell根本就不会检索与内置命令同名的/usr/bin/test命令或用户自己编写的test程序，除非明确指定必须执行/usr/bin目录中的test命令或自己提供的test程序。

（3）当执行一个Shell脚本时，系统首先需要打开这个文件，然后再逐行解释执行其中的命令语句。如果命令是一个编译的应用程序，则需要创建一个新进程，等到把程序装入内存后才能再执行，而Shell内置语句只需在现有的Shell进程中执行。

4. 管道中的命令顺序

在使用管道时，应适当地考虑过滤程序的位置顺序，逐步减少数据处理的信息量，以改善整个管道命令的运行效率。其中的一个原则是要尽可能地把过滤程序安排在最前面，把非过滤的程序安排在后面。尤其要注意把比较耗时的sort程序放在过滤程序的后面。grep命令就是一个较好的过滤程序的例子，可以滤掉大量不必要的数据，从而减少后续命令的数据处理量，提高整个管道命令的运行速度。因此，应像下面第一种方法（第一条语句）那样，尽可能地把grep等过滤程序放在前面，而不是后面。

```
who | grep something | sort
who | sort | grep something
```

第9章 用户管理

本章主要介绍Linux系统的用户管理，讨论怎样增加、修改或删除用户与用户组，介绍与用户管理有关的系统文件，说明如何设置用户的运行环境，详细讨论插件式认证模块，以及超级用户与sudo命令。

用户与用户组是Linux系统管理的一个重要部分，也是系统安全的基础。在Linux系统中，所有的文件、程序或正在运行的进程都从属于一个特定的用户。每个文件和程序都具有一定的访问权限，用于限制不同用户的访问行为。作为系统管理员，管理用户和用户组是一项重要的任务，有助于防止用户越权访问与其身份不相符的文件，执行系统任务，对系统造成破坏。总之，如果没有有效的用户管理，就无法保证系统的安全。

9.1 增加与删除用户

在Linux系统中，每个用户都具有一个唯一的身份标识，称做用户ID（或简称UID），以区别于其他用户。Linux系统按照一定的原则把用户划分为用户组，以便相关的同组用户之间能够共享文件。

一般地讲，Linux系统中的用户可以分为三类：超级用户（root）、管理用户和普通用户。但也可以把超级用户和管理用户通称为系统用户。

超级用户是一个特殊的用户（其用户标识号为0），拥有至高无上的访问权限，可以访问任何程序和文件。任何系统都会自动提供一个超级用户账号。在Ubuntu Linux系统中，为了确保系统的安全，超级用户账号通常是锁定的。Ubuntu Linux系统强烈建议，应尽量避免使用超级用户注册到系统中，如果确实需要执行系统管理与维护任务，可以在具体的命令前冠以sudo命令。

管理用户用于运行一定的系统服务程序，支持和维护相应的系统功能。这些用户的ID号位于1~999的范围之内。例如，ftp就是一个管理用户，用做FTP匿名用户，维护匿名用户的文件传输，同时提供匿名用户的默认主目录。

除了系统用户之外，其他均为普通用户。在访问Linux系统之前，每个用户都需要拥有一个用户账号。因此，系统管理员需要为每一个普通用户事先分配一个注册账号，把用户名及其他有关信息加到系统中。在利用用户名和密码成功地注册之后，用户才能访问系统提供的资源和服务，执行系统命令，开发和运行应用程序，以及访问数据库等。

对于自己的文件，用户均拥有绝对的权力，可以赋予自己、同组用户或其他用户访问文件的权限。例如，允许同组用户共享自己的文件，其他用户只能阅读或执行，但不能写文件等。

9.1.1 passwd文件

在安装Linux系统之后，系统已经事先创建了若干系统用户账号，其中包括超级用户root和管理用户daemon、bin和sys等，用于执行不同类型的系统管理和日常维护任务。

Linux系统中的用户账号信息是由/etc/passwd和/etc/shadow文件共同维护的。在这两个文件

中，每个用户都有一个相应的记录。当利用控制台等终端注册，按照系统的提示输入用户名和密码之后，系统将会根据用户提供的用户名检查/etc/passwd文件，然后根据用户提供的密码，利用同一加密算法加密后再与/etc/shadow文件中的密码字段进行比较，同时检查其他诸如密码有效期等字段。如果通过了验证，按照passwd文件指定的主目录和命令解释程序，用户即可进入自己的主目录，通过命令解释程序等界面访问Linux系统。

除了用户名和密码之外，每个用户还具有用户ID、用户组ID、主目录和命令解释程序等相关信息。这些信息分别存放在passwd和shadow文件中。

passwd文件包含了Linux系统中除密码之外的主要用户信息，每个用户信息占用一行，每一行由7个字段组成，字段之间以冒号“:”作为分隔符。passwd文件的格式定义如下：

```
username:password:uid:gid:comment:home_dir:login_shell
```

表9-1 给出了passwd文件中的每个字段及其简单说明。

表9-1 /etc/passwd文件

字段	简单说明
username	注册用户名。由2~8个字母（A~Z，a~z）和数字（0~9）、句点“.”、下画线“_”和连字符“-”等字符组成。在Linux系统中，用户名必须是唯一的，其中至少包含一个小写字母，且第一个字符应选用字母
password	这个字段原为用户密码，但密码实际上已移至/etc/shadow文件。如果用户设有密码，这个字段会包含一个小写字母“x”，加密后的密码存放在shadow文件中。如果这个字段为星号“*”，表示相应的用户无法正常地注册到系统
uid	用户ID（或称用户标识）。用户ID是系统或系统管理员分配给每个用户的一个唯一的数字标识，是系统识别每个用户的主要手段。当系统需要了解用户信息（如账号字段的内容）时，通常均以用户ID作为索引，检索passwd文件。用户ID是一个32位的无符号整数，每个ID必须位于0~65536的整数范围内。其中0~999保留为系统用户使用，自定义的用户ID应位于1000~65536的范围之内。考虑到与其他系统的兼容性，建议使用16位无符号整数的最大值32767作为用户ID的上限
gid	用户组ID（或称用户组标识）。Linux系统中的每个用户均属于某个用户组，每个用户组除有组名之外，还有一个相应的用户组ID。同样，ID号0~999保留为系统用户组使用
comment	注释信息。通常包含用户全名、电话号码和电子邮件地址等用户信息
home_dir	用户主目录（全路径名）。用户主目录是分配给用户的一个子目录，供用户存储个人文件用，也是用户注册后的起始工作目录。通常，Linux系统采用/home/username形式的用户主目录结构（环境变量HOME就是按这个字段设置的），其中username为注册的用户名
login_shell	指定用户注册后调用的Shell，也即命令解释程序。如果这个字段为空，则默认的命令解释程序为/bin/bash。用户也可以根据自己的爱好，选用其他命令解释程序，如Korn Shell（/bin/ksh）、zsh（/bin/zsh）或tcsh（/bin/tcsh）等

下面是Linux系统提供的初始用户信息（其中最后一个是安装系统时自定义的用户，也即sudo用户）：

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

```

sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
.....
gqxing:x:1000:1000:gqxing,,,:/home/gqxing:/bin/bash
$

```

9.1.2 shadow文件

/etc/shadow是一个限制普通用户访问的系统文件，其中存有加密形式的密码以及其他相关信息。与passwd文件相对应，shadow文件中的每个用户密码信息占用一行，每一行由9个字段组成，中间以冒号“:”作为分隔符。其文件格式定义如下：

```
username:password:lastchanged:mindays:maxdays:warn:inactive:expire:reserve
```

表9-2给出了shadow文件中的每个字段及其简单说明。

表9-2 /etc/shadow文件

字段	简单说明
username	用户注册名。参见/etc/passwd文件
password	加密形式的密码（通常是由crypt(3)函数生成的）。通常，用户输入的密码至少应包含6~8个字符，但生成的密码较长，由13~15个字符组成。如果加密后的密码以美元符号“\$”为起始字符，意味着加密的密码是由其他算法（非DES算法）生成的。例如，如果加密后的密码以“\$1\$”为起始字符串，意味着加密的密码是采用MD5之类的加密算法生成的。如果这个字段包含感叹号“!”或星号“*”字符，则表示用户无法正常地注册
lastchanged	从1970年1月1日开始算起，直至最后一次修改密码之日的天数
mindays	保持密码稳定不变的最小天数，仅当超过此限，才能修改密码。这个字段值必须大于等于零，才能启用密码有效期检查
maxdays	保持密码有效的最大天数。超过此限，系统将会强制提请用户更换新的密码
warn	指定在密码有效期到期之前需提前多少天向用户发出警告信息
inactive	指定在密码有效期到期之后一直不访问系统，但仍保证其账号信息有效的最多天数，超过此限将封锁用户账号。用户最后一次访问系统的信息记录于/var/log/lastlog文件中
expire	指定用户账号有效期的截止日期。到期后，账号将会自动失效，用户无法再注册到系统
reserve	保留字段

下面是Linux系统安装后的初始shadow文件：

```

$ sudo cat /etc/shadow
root:!:14562:0:99999:7:::
daemon*:14545:0:99999:7:::
bin*:14545:0:99999:7:::
sys*:14545:0:99999:7:::
sync*:14545:0:99999:7:::
.....
gqxing:$6$PodDPphah$XmVNLbVMuO28VGjg.....r101:14562:0:99999:7:::
$

```

9.1.3 用户管理实例



图9-1 “用户设置”对话框

在Ubuntu Linux系统中，为了管理用户和用户组，可在GNOME桌面中选择“系统→系统管理→用户和组”菜单，弹出如图9-1所示的对话框。单击“解锁”按钮，在弹出的“认证”对话框中输入sudo密码，即可开始增加新的用户或用户组，修改以及删除用户或用户组。注意，在使用GNOME桌面系统管理用户和用户组时，用户ID和用户组ID的默认起始编号为1001（安装Ubuntu Linux系统时创建的第一个用户占用了编号为1000的用户ID和用户组ID）。

在“用户设置”对话框中单击“添加用户”按钮，弹出如图9-2所示的“新建用户账户”对话框。此时可在“基本设置”一栏的“用户名”和“真实姓名”字段中分别输入用于系统注册的用户名和实际的用户名，对于“配置文件”字段，可选择默认的“Desktop user”，或从下拉菜单中选择“Administrator（赋予用户运行sudo命令的权限，但并非超级用户）”或“Unprivileged（表示普通用户）”，最后单击“用户权限”标签，切换到如图9-3所示的“用户权限”选项卡。



图9-2 “新建用户账户”对话框（1）



图9-3 “新建用户账户”对话框（2）

“用户权限”选项卡列出了可赋予用户的各种访问权限。选中左边的复选框，即可赋予相应权限（实际上是把用户加到各种用户组中，以便拥有相应的访问权限）。例如，如果赋予用户“管理用户”的权限，相当于把用户加到用户组admin，因而拥有运行sudo命令，执行特权命令的权限。

单击“高级”标签，切换到如图9-4所示的“高级”选项卡。在“高级设置”一栏中，可以设置用户的主目录、注册Shell、用户组以及用户ID。

在完成上述设置之后，单击“确定”按钮，返回如图9-5所示的“用户设置”对话框，可以看到已经把用户加到了系统中。

除了GNOME桌面环境，还可以使用Linux系统的基本命令，实现用户和用户组的管理与维护。下面以命令行方式为例，说明怎样增加、修改和删除用户。



图9-4 “新建用户账户”对话框（3）



图9-5 完成设置后的“用户设置”对话框

1. 增加新用户

在命令行方式下，为了增加用户，可以使用useradd命令，其语法格式如下：

```
useradd [-u uid] [-g group] [-d home_dir] [-s shell] [-c comment]
        [-m [-k skel_dir]] [-N] [-f inactive] [-e expire] login
```

其中，login表示新增用户的注册用户名。其他选项的说明如表9-3所示。

表9-3 useradd命令的常用选项

选项	GNU选项	简单说明
-u uid	--uid uid	用于指定新增用户的用户ID。通常，用户ID是当前已经分配的最大ID号加1。例如，如果当前已分配的用户ID为1000、1005和2000，则默认的下个可用用户ID将是2001
-g group	--gid group	用于指定一个现有用户组的用户组ID或用户组名
-N	--no-user-group	通常，增加新用户时会同时增加一个与用户名同名的用户组。利用这个选项可以禁止创建同名的用户组。把用户加到/etc/default/useradd文件指定的默认用户组（users）中
-d home_dir	--home home_dir	用于指定新增用户的主目录。通常，用户主目录为/home/username。其中，username为新增用户的注册用户名，也即命令语法格式中的“login”参数
-s shell	--shell shell	用于指定命令解释程序Shell的完整路径名。默认的命令解释程序为/bin/bash
-c comment	--comment comment	用于指定用户全名、电话号码以及电子邮件地址等注释信息
-m	--create-home	在增加新用户时，如果用户的主目录不存在，则创建用户主目录。同时，把/etc/skel目录或“-k”选项指定目录中的初始化文件复制到用户主目录中
-k skel_dir	--skel skel_dir	用于指定存储用户初始化文件（如profile）的目录，以便useradd命令能够把其中的文件复制到用户主目录。通常，系统将会提供一组标准的初始化文件，为用户设置最基本的运行环境。用户也可以此为起点，增加自己的设置。系统默认的目录位置为/etc/skel

（续表）

选项	GNU选项	简单说明
-f <i>inactive</i>	--inactive <i>inactive</i>	用于指定相应用户一直未访问系统，但仍保证其注册账号信息有效的最多天数。超过此限将锁住用户账号
-e <i>expire</i>	--expiredate <i>expire</i>	指定注册用户的有效期，也即截止日期。如果超过指定的日期，则不允许使用相应的用户名访问系统。有效的日期格式为“YYYY-MM-DD”。这个选项主要用于增加一个临时用户

例如，为了增加一个用户cathy，同时增加一个用户组cathy，可以使用下列命令：

```
$ sudo useradd -u 1001 -d /home/cathy -m -s /bin/bash cathy
[sudo] password for gqxing:
$
```

执行上述命令之后，将会在/etc/passwd、/etc/shadow和/etc/group文件中各增加一行新用户cathy的有关信息：

```
$ grep cathy /etc/passwd
cathy:x:1001:1001::/home/cathy:/bin/bash
$ sudo grep cathy /etc/shadow
cathy!:14583:0:99999:7:::
$ grep cathy /etc/group
cathy:x:1001:
$
```

采用useradd命令增加用户之后，还需要使用passwd命令为用户设置一个临时密码，否则无法正常注册。用户注册之后，可以再使用passwd命令修改自己的密码：

```
$ sudo passwd cathy
Enter new UNIX passwd:
Retype new UNIX passwd:
passwd: password updated successfully
$
```

在选用用户名、用户ID和用户组ID时通常应遵循下列原则：

- 用户名应包含2～8个字母和数字，第一个字符必须是字母，而且至少有一个字符是小写字母。注意，即使用户名中允许包含句点“.”、下画线“_”或连字符“-”，也不建议使用。
- 避免使用/etc/passwd和/etc/group文件中已有的用户名、用户ID、用户组名以及用户组ID。此外，用户ID和用户组ID号0～999是系统保留的，普通用户不应使用。

一旦创建了用户账号，就可以使用passwd命令设置密码。使用usermod等命令修改passwd和shadow文件，更改用户的其他相关属性。



在增加、修改和删除用户时，如果采用编辑器，手工修改passwd和shadow文件，必须考虑以下因素：

- 用户属于哪个用户组，相应的用户组是否存在。
- 除了修改passwd文件之外，还要在/etc/shadow文件中增加、修改和删除相应的用户项。
- 创建和删除用户主目录。

- 此外还要创建或复制用户初始化文件，如profile和bashrc等文件（取决于选定的命令解释程序）。

2. 修改用户账号信息

除非用户名或用户ID与其他用户冲突，一般情况下不要轻易修改用户账号中的用户名和用户ID，因为这将涉及到用户已经创建的所有文件和目录。否则，需要同时修改用户所有文件和目录的文件属主等属性。但可以放心地修改用户账号的其他字段：

- 注释字段；
- 命令解释程序；
- 密码及密码的其他属性（shadow文件）；
- 主目录及主目录的访问权限。

修改用户信息时，可以利用编辑器，以手工方式直接编辑passwd和shadow文件，也可以使用usermod命令。usermod命令的语法格式如下：

```
usermod [-u uid] [-g group] [-d home_dir] [-s shell] [-c comment] [-f inactive]
        [-e expire] [-l new_logname] login
```

其中，“-l”选项用于指定一个新的注册用户名。其他选项的说明参见表9-3。

例如，为了把用户cathy的命令解释程序bash更换为ksh，可以使用下列命令：

```
$ sudo usermod -s /bin/ksh cathy
$
```



注意 除了超级用户root的密码之外，不要试图修改和删除系统用户账号的任何信息。

3. 删除用户

一旦用户不再需要访问Linux系统，理应从系统中删除其账号信息。删除用户账号时可以使用userdel命令，也可以采用手工编辑的方式。使用userdel命令的好处是，只需一个命令即可删除passwd、shadow和group文件中的相应用户和用户组信息，同时还会把用户主目录中的所有文件和目录一同删除。如果采用手工方式，既需要编辑passwd、shadow或group文件，还需要删除用户主目录及其中的文件。userdel命令的语法格式如下：

```
userdel [-r] login
```

其中，“-r”选项意味着同时从系统中删除用户的主目录，包括其中的文件和子目录。

4. 封锁用户账号

有时，也许需要临时或永久禁止或者封锁某个用户账号，禁止其注册。为此，可以使用passwd命令，或者直接修改/etc/shadow文件，在密码字段前面增加一个感叹号“!”前缀，或把密码字段改为星号“*”字符，以封锁相应的用户账号，从而防止用户再利用此账号注册到系统。另外一种方法是，利用usermod命令修改shadow文件中的expire字段，把其中的日期改成一个过时的日期。

passwd命令的语法格式如下：

```
passwd [-adehlSu] [-i inactive] [-m min] [-w warn] [-x max] [login]
```

表9-4 给出了passwd命令的部分选项及其说明（除非特别说明，相应的选项仅限于超级用户使用）。

表9-4 passwd命令的部分选项

选项	GNU选项	简单说明
-a	--all	这个选项只能与“-S”选项一起使用，以查询所有用户账号的状态信息
-d	--delete	删除指定用户的密码，同时在passwd文件的password字段中增加一个星号“*”字符，以封锁指定用户的注册账号，使用户无法再注册到系统
-e	--expire	令指定用户的密码立即失效，强制用户在下一次注册时立即更换密码
-i inactive	--inactive inactive	指定在密码有效期到期之后一直不访问系统，但仍保证其账号信息有效的最多天数，超过此限将封锁用户账号（实际上相当于设置shadow文件中的inactive字段）
-l	--lock	封锁指定用户的密码（在加密形式的密码字段中增加一个感叹号“!”前缀），从而封锁指定用户的注册账号，拒绝相应用户再次注册。当某用户离开公司，有关文件尚未完全交接时，可以使用此选项先封锁用户账号，待清理完毕之后再删除用户账号信息
-m min	--mindays min	指定密码保持不变的期限——即在更换密码之前必须保持密码不变的最少天数。换言之，即指定多少天之内不能更改密码（相当于设置shadow文件中的mindays字段）
-S	--status	显示指定用户的密码状态属性
-u	--unlock	解除被封锁的用户注册账号（删除密码字段前面的感叹号“!”前缀）
-w warn	--warndays warn	指定在密码有效期到期之前的多少天向用户发出警告信息（相当于设置shadow文件中的warn字段）。如果禁止检查密码有效期（shadow文件中的mindays字段为0），这个选项没有意义。默认情况下是有效期截止日的前一天
-x max	--maxdays max	指定密码的最大有效期限，以天数计算（相当于设置shadow文件中的maxdays字段）。如果禁止检查密码有效期（shadow文件中的mindays字段为0），这个选项（或字段）没有意义

例如，为了封锁guest的用户账号，可以使用下列命令：

```
$ sudo passwd -l guest
passwd: password expiry information changed.
$
```

5. 定期更改密码

普通用户可以使用不带任何选项和参数的passwd命令修改自己的密码，而超级用户可以利用passwd命令的大量选项及参数维护系统中的用户账号信息。例如，利用密码的有效期设置，可以强制用户定期修改自己的密码。

例如，为了强制用户hwang下一次注册时立即更换密码，可以使用下列命令：

```
$ sudo passwd -e hwang
passwd: password expiry information changed.
$
```

为了查询用户cathy的密码属性，可以使用下列命令：

```
$ sudo passwd -S cathy
cathy P 11/15/2009 0 99999 7 -1
$
```

在上述输出信息中，P表示用户cathy已经设置了密码。其他密码属性包括NP，表示未设置密码，L表示用户账号已经封锁。

6. 检验用户的有效用户ID

为了检查当前用户的有效用户ID，可以使用id命令。例如，当cathy注册到系统之后，id命令的输出结果如下：

```
$ id
uid=1001(cathy) gid=1001(cathy) groups=1001(cathy)
$
```

使用su命令能够改变用户的有效用户ID，而不必另行注册。在输入其他用户名和密码之后，一个用户可以“合法地变成”另一个用户。除此之外，如果在su命令之后加上连字符“-”、“-l”或“-login”选项，还可以直接进入其他用户的主目录，如同使用其他用户名和密码直接注册一样。否则，即使改变用户身份，仍会保持之前的工作目录不变。例如：

```
$ su cathy
Password:
$ pwd
/home/gqxing
$ exit
exit
$ su - cathy
Password:
$ pwd
/home/cathy
$
```

9.2 定制用户的工作环境

除了分配用户名和用户ID，设定用户主目录，供用户创建和存储文件之外，还需要有一个工作环境，使用户能够借助系统提供的各种工具和资源访问系统，开发和运行自己的应用。在注册到系统之后，用户的工作环境是由选定的命令解释程序和相应的用户初始化文件确定的。因此，用户管理的另外一个任务是选择作为用户界面的命令解释程序和用户初始化文件。

9.2.1 选择命令解释程序

Linux系统配备的标准命令解释程序是Bash (bash)，但同时也支持Korn Shell (ksh)、TCShell (tcsh) 以及ZShell (zsh) 等，用户可以自由选用。每个Shell各具特色，根据自己的需要，用户可以选用不同的命令解释程序。/etc/shells文件中列出了Ubuntu Linux系统支持的所有命令解释程序（注意，ksh、tcsh及zsh等需要单独下载）：

```
$ cat /etc/shells
# /etc/shells: valid login shells
```

```
/bin/csh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
.....
/bin/sh（从6.10版开始，/bin/sh已由bash改为dash的符号链接文件。注意，这两者是不兼容的）
/bin/dash
/bin/bash
.....
$
```

下面简单介绍一下4个应用比较广泛，且最具代表性的Shell，供用户在选用命令解释程序时参考。

1. Bourne Again Shell

Bash（Bourne Again Shell）是基于POSIX1003.2标准开发的一个免费版的Shell，与Borne Shell和Korn Shell一脉相承，在功能上又有较大的提高。Bash兼收并蓄，广泛采纳了其他Shell的优点，支持emacs与vi两种命令行编辑功能，支持命令历史、命令别名和作业控制等机制，改善了人机交互能力，具有良好的用户界面和工作环境。作为基本配置，Bash是Linux系统首选的Shell，其地位就像Bourne Shell之于UNIX。

2. Korn Shell

Korn Shell（/bin/ksh）是UNIX系统中继Borne Shell与CShell之后推出的第三个著名的Shell。Korn Shell以Borne Shell为基础，同时充分吸纳了CShell的优点，兼具Bourne Shell的功能与CShell的交互能力，支持命令行编辑、命令历史、命令别名和作业控制等功能，极大地增强和丰富了Bourne Shell的基本功能，并与Borne Shell完全兼容，同时也奠定了现代Shell（如Bash）的基础。

3. TC Shell

CShell原为BSD版UNIX系统的命令解释程序，由加州大学伯克利分校计算机系的Bill Joy开发。TCShell（/bin/tcsh）是在CShell的基础上开发的，继承并发展了CShell的全部功能特性，支持emacs命令行编辑、命令历史、命令别名和作业控制等机制，改善了人机交互能力，具有良好的用户界面和工作环境。但需要注意的是，TCShell与Bash和Korn Shell等Bourne系列的Shell是不兼容的，两个系列Shell环境中开发的脚本不能互换使用。

4. Z Shell

同上述其他Shell一样，zsh既是一个交互式的命令解释程序，也是一个强有力的编程语言。zsh吸收并集成了bash、ksh以及tcsh等Shell的许多功能特性，能够提高用户与Linux系统交互的效率，非常适合用做交互式Shell。可以说，zsh是上述Shell的超集。zsh自己也增加了许多先进的功能，这些功能位于动态库中，可根据具体需要加载或卸载，以节省内存。

任何时刻，每个用户只能使用一个命令解释程序，但可以随时从一个Shell切换到另外一个Shell环境。为了确定当前究竟处于哪一个Shell环境，可以使用“ps -f”命令，找出ps的父进程，即可确定当前使用的是哪一个Shell，例如：

```
$ ps -f
UID      PID  PPID  C STIME TTY          TIME CMD
gqxing   1586  1531   0 16:18 pts/0        00:00:00 -bash
gqxing   2365  1586   0 18:45 pts/0        00:00:00 ps -f
$
```

增加用户时，每个用户都可以一次性地选定一个命令解释程序，如果未做出选择，系统默认的命令解释程序为Bash。如果需要，也可以通过usermod等命令修改用户账号信息，或直接修改passwd文件，随时更换命令解释程序。一经注册，即可调用指定或默认的命令解释程序。

Shell是大多数用户访问Linux系统的主要命令行界面，Shell功能的任何改进，如命令行编辑、命令或文件名补充以及命令提示符定制等，都有助于提高用户访问Linux系统的效率。Shell的功能和特点是用户选择Shell的依据。

表9-5给出的Shell功能特性的简单比较，可供用户选择Shell时参考。至于究竟选用哪一个Shell，还要取决于用户的爱好和习惯。

表9-5 Shell功能特性比较

Shell的主要特性	bash	ksh	tsh	zsh
是否用做Linux系统的标准Shell	Y	N	N	N
作业控制	Y	Y	Y	Y
命令历史	Y	Y	Y	Y
命令别名	Y	Y	Y	Y
命令行编辑	Y	Y	Y	Y
vi命令行编辑	Y	Y	Y	Y
emacs命令行编辑	Y	Y	Y	Y
文件覆盖保护（noclobber）	Y	Y	Y	Y
Shell函数	Y	Y	N	Y
命令或文件名补充	Y	Y	Y	Y
用户名补充	Y	Y	Y	Y
主机名补充	Y	Y	Y	Y
命令历史补充	Y	N	Y	Y
拼写校正	N	N	Y	Y
进程替换	Y	N	N	Y
是否易于定制命令提示符	Y	Y	Y	Y
是否具有完整的信号捕捉与处理机制	Y	Y	N	Y
内置的数学运算	Y	Y	Y	Y
是否支持较大的参数列表	Y	Y	Y	Y

9.2.2 设置用户初始化文件

初始化文件（也称启动文件）是一种Shell脚本，其中包含用户注册后必须执行的各种命令，以便在注册之后，能够设置用户的工作环境。原则上讲，初始化文件与普通Shell脚本一样，均

可执行任何处理任务。但是，初始化文件的主要任务是设定用户的工作环境。

初始化文件分为系统初始化文件和用户初始化文件。系统初始化文件主要用于设置系统范围的用户工作环境，如设置命令检索路径、用户主目录、命令提示符以及终端类型等环境变量。用户初始化文件主要用于进一步定制自己的工作环境，如设置命令别名、个性化的命令提示符以及命令检索路径等。

每个Shell都提供并支持各自的初始化文件，其中包括位于/etc目录的系统初始化文件，如/etc/profile文件，以及位于用户主目录的用户初始化文件，如\$HOME/.profile文件等。表9-6给出了各种Shell支持的用户初始化文件。

表9-6 Shell支持的用户初始化文件

Shell	初始化文件	简单说明
Bash	\$HOME/.profile	用于设置用户自己的工作环境，如设置PATH变量，定义命令别名等作用同上，但Ubuntu Linux系统并不使用这个用户初始化文件
	\$HOME/.bash_profile	作用同上，但Ubuntu Linux系统并不使用这个用户初始化文件
	\$HOME/.bash_login	用于设置交互式非注册Shell的用户工作环境
	\$HOME/.bashrc	用于定义退出Bash时需要执行的善后处理任务，如清除临时文件等
	\$HOME/.bash_logout	
Korn Shell	\$HOME/.profile	用于设置用户自己的工作环境，如设置PATH变量，定义命令别名等
TC Shell	\$HOME/.tcshrc	用于设置用户自己的工作环境，如设置环境变量，定义命令别名等
	\$HOME/.cshrc	同上
	\$HOME/.login	用于定义每次注册时需要执行的命令，如设置环境变量，定义终端类型等
	\$HOME/.logout	用于定义退出TC Shell时需要执行的善后处理任务，如清除自己的临时文件等
Z Shell	\$HOME/.zshenv	用户初始化文件，用于设置用户自己的工作环境，如设置命令检索路径等环境变量
	\$HOME/.zprofile	用户初始化文件，用于设置用户自己的工作环境，如设置命令别名和函数等
	\$HOME/.zshrc	用户初始化文件，用于设置用户自己的工作环境，如设置命令历史文件等
	\$HOME/.zlogin	用户初始化文件，用于设置用户自己的工作环境，如设置终端类型等
	\$HOME/.zlogout	用于定义退出Z Shell时需要执行的善后处理任务，如清除自己的临时文件等

如果选用的是Bash，当用户注册或打开一个新的终端窗口时，系统首先会运行系统初始化文件/etc/profile。用户可以查阅系统初始化文件，但不应修改其中的任何内容，以免影响其他用户。之后，系统还会检测并执行相应的用户初始化文件，以定制用户自己的运行环境。

每个Linux系统通常都提供若干标准的用户初始化文件，其中的默认设置基本上都能够满足用户的常规要求。如有特殊需求，如定制系统提示符，设置命令别名等，只需在此基础上进行适当的修改，增加特定的变量设置或命令即可。

下面是Ubuntu Linux系统提供的用户初始化模板文件：

- /etc/skel/bash_logout
- /etc/skel/bashrc
- /etc/skel/profile

在利用`useradd`命令增加新的用户账号时, 如果使用“-k”和“-m”选项指定了`/etc/skel`目录, `useradd`将会把`/etc/skel/`目录中的所有文件复制到用户的主目录。此时, 应根据选用的命令解释程序, 删除不必要的文件(如果存在)。然后, 利用`.profile`文件作为用户初始化文件的起点, 定制自己的Shell工作环境。

在Ubuntu Linux系统的Bash运行环境中, 用户需要设置和使用的通常只有`~/profile`和`~/bashrc`两个初始化文件。如果想要增加自己的设置, 应尽量加在`~/profile`文件中, 因为这个文件仅在注册时执行一次, 但`~/bashrc`文件有可能多次执行, 因而会影响系统的性能。

9.2.3 定制Shell工作环境

1. Shell运行环境

每个Shell都会维护一组由login程序、系统用户初始化文件及用户初始化文件定义的各种变量。其中包括:

- 环境变量: 可用于Shell调度执行的所有进程的变量称为环境变量。使用`env`或`set`命令可以查阅这些变量的设置情况。
- Shell本地变量: 仅对当前Shell有效的变量称为本地变量。在TCShell中, 一组Shell本地变量(如`user`、`term`、`home`和`path`)与相应的环境变量具有特殊的关系, 二者之间互相影响。

在Bash中, 本地变量和环境变量通常均统一采用大写字母(但并不禁止使用小写字母和其他字符)。如果期望变量的设置能够用于随后调度执行的任何命令或程序, 必须使用`export`命令公布用户自己设置的变量。

在用户初始化文件中, 可以修改预定义变量的值, 也可以设置附加的变量, 定制自己的Shell工作环境。如果想在Bash和Korn Shell的用户初始化文件中设置变量, 可以使用下列命令:

```
VARIABLE=value; export VARIABLE
```

例如, 为了使vim编辑器等能够正常工作, 需要根据自己使用的具体终端类型, 采用“`TERM=termtype; export TERM`”的命令形式定义终端, 以便系统能够检索TERMINFO数据库, 确定终端的特性。有关变量的更多信息, 参见第7章“Shell基础知识”。

2. 设置环境变量

在数据库等应用程序中, 需要用到许多环境变量。一些软件厂商通常也都提供一个环境变量设置脚本, 供用户设置运行环境。

为使环境变量的设置能够用于当前的Shell运行环境, 可以采用两种方法: 一是利用“.”或`source`命令读取并执行Shell脚本, 在当前的Shell中设置运行环境; 二是把环境变量的设置语句加到`.profile`等用户初始化文件中。

“.”或`source`命令的主要功能是在当前Shell的控制下, 读取指定的Shell脚本文件, 由当前的Shell解释执行。“.”或`source`命令的调用方法如下:

```
. script
```

或

```
source script
```


其中，script是一个Shell脚本文件，文件中包含必要的环境变量设置语句。

使用“.”或source命令的主要目的是执行指定脚本文件中的命令，在当前的Shell中设置环境变量。例如，在使用MySQL数据库之前，用户也许需要运行诸如“mysql_env.sh”的环境变量设置脚本，或者在profile文件中增加下列内容，从而设置自己的运行环境：

```
MYSQL_HOST=iscas
MYSQL_TCP_PORT=3306
TMPDIR=/tmp
export MYSQL_HOST MYSQL_TCP_PORT TMPDIR
```

假定mysql_env.sh是一个环境变量设置脚本，可使用“.”或source命令读取并运行mysql_env.sh脚本文件中的语句，在当前的Shell中设置环境变量。即使退出脚本，环境变量的设置仍将保持有效。

```
$ . ./mysql_env.sh
$
```

如果使用下列命令，则在退出子Shell或Shell脚本之后，当前的Shell环境不会留存脚本中设置的任何变量。

```
$ sh mysql_env.sh
```

或

```
$ ./mysql_env.sh
```

3. 设置命令检索路径

设置环境变量的一个重要内容是设置PATH变量。Shell变量PATH用于设置命令的检索路径，定义命令所在的目录。PATH变量包含一系列目录，目录名之间由冒号“:”分隔。目录的列举顺序确定了命令的检索顺序。为了提高系统的响应速度，常用命令所在的目录应放在靠前的位置。

假定环境变量PATH设置为“/usr/bin:/usr/sbin:”，意味着命令的检索路径依次为/usr/bin、/usr/sbin和当前目录。指定当前目录至少有三种方法：一是使用两个相邻的冒号，二是在原有路径名的前部或后部增加一个冒号“:”，三是增加一个句点“.”。

当用户采用完整的绝对路径提交命令（输入的命令名以斜杠“/”开始）时，Shell将会省略PATH变量检索步骤，按照给定的路径检索命令，直接加载并执行指定的命令。否则，当用户仅输入命令名本身时，Shell将会根据PATH变量指定的目录顺序，依次检索与之匹配的命令文件。

不管在哪一个目录位置，如果发现了匹配的命令文件，文件的访问权限是可执行的，而且也不是目录文件时，则创建一个新的进程，执行该命令。如果指定的命令既非绝对路径，从PATH变量列举的目录中也找不到匹配的命令文件，系统将会输出下列错误信息：

```
bash: cmd: command not found
```

上述错误信息表明：或者命令名的输入有误；或者给定的命令确实不在PATH变量指定的检索目录中。对于第二种情况，校正的办法之一是使用绝对路径名，之二是修改PATH变量，把命令所在的目录加到PATH变量中，如附加到PATH变量值的后面。

在安装第三方的软件时，通常需要修改PATH变量，以免出现“命令不存在”或使用绝对路径名运行命令的麻烦。

用户初始化文件通常已经设定了PATH变量，包括命令路径的检索顺序。但是，大多数用户均会修改这个变量，增加自己的命令检索路径，或调整命令路径的检索顺序。例如，为了把当前目录作为PATH变量中的一个命令检索路径，可以使用下列命令：

```
PATH=$PATH:.; export PATH
```



如果PATH变量的命令检索路径设置不当，有可能导致系统响应时间过长，也可能导致系统执行不正确的命令。为了提高PATH变量检索的有效性，下面是设置目录检索路径的若干建议：

- 如果想加入当前工作目录，通常应把当前工作目录作为最后一个检索路径。如果安全并非主要问题，可以把当前工作目录作为第一个检索路径；
- 应尽可能地缩短命令检索路径，以减少响应时间，提高系统的性能；
- 命令的检索顺序是从左到右，经常使用的命令，其所在目录应位于检索路径的最前面；
- 确保命令检索路径不包含重复的目录，以免执行重复的检索；
- 应尽量避免检索大的目录，如果可能，应把大的目录置于检索路径的最后面；
- 如果存在，应把本地目录置于远程目录前端，以减少出现系统挂起的机会，也减轻不必要的网络负荷。

为了定制用户自己的命令检索路径，使之能够首先检索系统目录，同时把自己的特定目录及当前工作目录也包括在检索路径中，可在用户初始化文件中增加下列内容：

```
PATH=$PATH:$HOME/bin:
export PATH
```

4. 设定语言环境

LANG和LC系列环境变量用于设置特定的语言环境(locale)，包括系统提示信息、格式转换、应用惯例和表达方式、字符排序原则，以及日期、时间、货币和数值的输出形式等。

通常，只需设置LANG环境变量，即可确定一个语言环境。实际上，在设置LANG环境变量时，系统也能够连带设置所有的LC系列变量(LC_ALL变量除外)，因而能够提供正确的语言环境。同样，设置LC_ALL变量也能同时设置其他LC系列变量，但LANG与LC_ALL两个变量互不影响。此外，也可以单独设置下列LC系列变量，以设置部分语言环境。

- LC_COLLATE (指定字符排序原则)
- LC_CTYPE (指定语言类型，包括字符类型、字符转换与多字节字符的宽度等)
- LC_MESSAGES (指定系统提示信息采用的语言)
- LC_NUMERIC (指定小数点与千分位的表达方式)
- LC_MONETARY (指定货币符号等)
- LC_TIME (指定日期与时间的表示方式)
- LC_ALL (意味着设定上述的所有LC*变量)

Ubuntu Linux系统采用POSIX格式“xx_YY.CHARSET”定义语言环境。其中，“xx”是ISO-639标准定义的语言代码，“YY”是ISO-3166标准定义的国家或地区代码，“CHARSET”是/usr/share/i18n/locales目录中列举的字符集之一。注意，Ubuntu Linux系统仅支持Unicode（UTF-8）字符集。

为了利用环境变量设置简体中文语言环境，可以把LANG变量设置为zh_CN.UTF-8或zh_CN.utf8。因此，可在用户初始化文件中增加下列内容：

```
LANG=zh_CN.UTF-8; export LANG
```

为了查询Linux系统支持的语言环境，可以运行“locale -a”命令。欲了解当前设定的语言环境，可以运行下列locale命令：

```
$ locale
LANG=zh_CN.UTF-8
LANGUAGE=zh_CN.UTF-8
LC_CTYPE="zh_CN.UTF-8"
LC_NUMERIC="zh_CN.UTF-8"
LC_TIME="zh_CN.UTF-8"
LC_COLLATE="zh_CN.UTF-8"
LC_MONETARY="zh_CN.UTF-8"
LC_MESSAGES="zh_CN.UTF-8"
.....
LC_ALL=
$
```

如果想要在中英文语言环境之间切换，可把环境变量LANG交替地设置成zh_CN.utf8或C。

5. 定制命令提示符

当以命令行方式注册到Linux系统，或在GNOME桌面打开一个终端窗口时，窗口的左边将会出现Shell命令提示符，说明调用的Shell已经就绪，用户可以输入命令。通常，Linux系统的提示符为“[\u@\h \W]\\$”。其中，“\u”表示注册的用户名，“\h”表示系统的主机名，

“\W”表示当前工作目录最后一个子目录的名字，“\\$”表示根据用户的身份显示“\$”（普通用户）或“#”（超级用户）命令提示符。

例如，当使用gqxing注册时，其命令提示符如下（为了使命令的输出清晰起见，本书的示例中均省略了方括号及其中的内容，特此说明）：

```
[gqxing@iscas ~]$
```

Linux系统的命令提示符共分为四级，依次由Shell的内置变量PS1、PS2、PS3和PS4分别定义。当输入的命令不完整，输入的引号未成对出现，或直接在命令行上输入Shell编程语句时才会用到第二级命令提示符。PS3和PS4分别用于select循环结构和Shell脚本调试。

最常见的命令提示符主要是第一级命令提示符。在Bash中，命令提示符能够给出丰富的提示信息，有助于用户输入命令。对于资深用户来讲，系统默认的命令提示符在提供较多信息的同时，有时也会显得冗长，且容易与命令的输出混杂在一起。为了定制自己的命令提示符，实际上只须参照PS1变量的设置（PS1='[\u@\h \W]\\$'），修改PS1变量的定义，即可达到改变命令提示符的目的。

表9-7中给出了设置Bash命令提示符时可用的部分特殊字符，这些特殊字符可用于显示注册的用户名、系统的主机名或当前的工作目录等信息，用户可根据自己的喜好予以选用。

表9-7 Bash命令提示符中可用的部分特殊字符

字符	简单说明
\\$	输出表示用户身份的提示符。超级用户为“#”，普通用户为“\$”
\!	显示当前命令在命令历史记录中的序号
\#	显示当前命令自调用Shell起的序号。从1开始编号
\a	输出提示音
\d	按照“星期 月 日”的格式显示日期，如“Mon May 18”
\h	显示系统的主机名（不包括域名）
\H	显示系统的规范主机名（包括域名）
\j	显示后台作业的数量（包括当前正在运行或暂停的作业）
\l	显示终端的设备名
\n	在提示信息中插入一个换行字符
\r	在提示信息中插入一个回车字符（相当于清除回车字符之前的所有提示信息）
\s	显示当前使用的Shell
\@	以12小时“AM/PM”的格式显示时间，如“03:30 PM”或“10:12 上午”
\T	以12小时“HH:MM:SS”的格式显示时间，如上述的“03:30 PM”将会显示为“03:30:00”，没有AM和PM之区分
\t	以24小时“HH:MM:SS”的格式显示时间，如“15:30:00”
\A	以24小时“HH:MM”的格式显示时间，如“15:30”
\u	显示当前的注册用户名
\w, \W	显示Bash的版本信息
\w	显示当前工作目录的完整路径名
\W	显示当前工作目录的最后一个子目录名
\	在命令提示符中插入一个反斜杠
\[, \]	“\[”标志一个不可打印字符序列的开始，以便在命令提示符中嵌入一个终端控制序列，“\]”表示不可打印字符序列的结束

但在Korn Shell中，命令提示符只是简单的注释符号“#”或美元符号“\$”。为了改变这种单调的命令提示符，也可以简单地修改PS1变量的定义。例如，假定希望命令提示符中能够包含命令序号和当前工作目录，可以采用下列设置方法：

```
$ PS1='[! `pwd`]$ '
[160 ~]$
```

当输入了部分命令，或误输入的引号未成对出现而按下Enter键时，Shell将会输出第二级命令提示符（通常为大于号“>”）。出现这一情况时，解决的办法一是继续完成命令的输入，二是利用Ctrl-C组合键终止误输入的命令，使Shell能够继续处理新的命令。为了把默认的第二级命令提示符改为more，可以使用下列命令：

```
$ PS2='more ? '
$
```

6. 定义命令别名

命令别名的定义仅在当前的Shell中有效，一旦退出Shell，所有的命令别名定义也将随之消

失。为了设置永久性的命令别名，可在\$HOME/.profile文件中定义，不管何时注册到Linux系统，随时都可以直接使用。

正如第5章“文件与目录操作”所述，cp、rm或mv等命令存在误删或覆盖原有同名文件的问题。为了防止此类情况出现，可将下列命令别名加到自己的profile或bashrc文件中（为了彻底解决这个问题，也可以由系统管理员一次性地修改/etc/skel目录中的profile或bashrc文件，把下列命令别名定义加到其中）：

```
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```



每当启动Shell时，Shell都会读取并执行profile文件。如果定义的命令别名过多，Shell的启动过程将会减慢。因此，一般不要定义太多的命令别名。

7. 定制用户初始化文件

下面以Bash为例，说明怎样以标准的用户初始化文件为基础，定制自己的用户初始化文件。通常，系统提供的标准profile文件内容如下：

```
$ cat $HOME/.profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
$
```

为了定制自己的运行环境，修改Shell的检索路径，使之包含用户主目录及当前目录，设置中文语言环境，以vi/vim作为命令行编辑器等，可以利用任何编辑器，修改profile，增加下列内容：

```
PATH=$PATH:$HOME/bin:.
LANG=zh_CN.utf8
EDITOR=vi
export PATH LANG EDITOR
```

```
alias r='fc -e -'  
alias rm='rm -i'  
alias cp='cp -i'  
alias mv='mv -i'  
.....
```

9.3 增加与删除用户组

为了使相关的用户能够访问共同的资源与服务，Linux系统支持用户组的概念。Linux系统中的每个用户都从属于某个用户组，同组的用户具有相同的用户组访问权限。此外，每个用户还可以临时改换其有效用户组，以便与其他用户组相关联。这种灵活性意味着用户除了可以按照工作性质或所在部门（如产品部或技术支持部）从属于一个主要用户组，还可以根据工作的需要（如项目组），与其他用户组共享数据与访问权限。

在Ubuntu Linux系统中，最典型的一个例子就是，如果一个用户从属于用户组admin，即可运行sudo命令，执行系统管理与维护方面的系统命令。

在Linux系统中，与用户组有关的所有信息通常均存诸在/etc/group系统文件中（参见表9-8）。文件中的每一行定义一个用户组，其语法格式如下：

```
group_name:password:gid:user_list
```

表9-8 /etc/group文件

字段	简单说明
group_name	用户组的名字。用户组名可由2~8个字符组成。例如，可以定义一个用户组math，使数学系的用户均归属于math用户组
password	通常为“x”。这个字段目前并无实际的意义
gid	用户组ID。系统中的用户组ID必须是唯一的。同用户ID一样，用户组ID是一个32位的无符号整数。每个ID必须位于0~65536的整数范围内，其中0~999保留为系统用户组使用，自己定义的用户组ID应位于1000~65536的范围之内。考虑到与其他系统的兼容性，建议使用16位无符号整数的最大值32767作为用户组ID的上限
user_list	用户组成员列表。其中可以包含属于当前用户组的所有用户名，用户名之间需加逗号分隔符

下面是安装Ubuntu Linux系统之后系统提供的默认用户组信息：

```
$ cat /etc/group  
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:gqxing  
.....  
users:x:100:  
.....  
admin:x:115:gqxing  
.....  
gqxing:x:1000:  
$
```

为了在系统中增加新的用户组，可以手动编辑/etc/group文件，也可以使用groupadd命令。groupadd命令的语法格式如下：

```
groupadd [-g gid [-o]] group
```

其中，“-g”选项用于指定新增的用户组ID。如果忽略此选项，默认的用户组ID为已分配的最大ID号加1。例如，如果用户组ID号1000、1005和2000已经分配，则默认的用户组ID为2001。

如果想要增加一个名为banking，用户组ID为2000的用户组，可以使用下列命令：

```
$ sudo groupadd -g 2000 banking
$
```

然后，可以使用grep、cat命令或其他工具验证用户组是否已经加到系统中：

```
$ grep banking /etc/group
banking:x:2000:
$
```

如果想要修改或删除用户组，可以分别使用groupmod和groupdel命令。因为用户组的修改和删除比较简单，故此处不再赘述。在管理和维护用户组时，最简单的方法其实是利用编辑器，手动编辑/etc/group文件。

9.4 监控用户

用户管理的一个重要目的就是保证用户的活动总是处于一个正常的范围之内。如果某个用户正在过度地占用CPU资源，则可以使用nice命令降低用户进程的优先级别，参见第10章“进程管理”。从系统安全的角度考虑，监控用户的活动可能会发现一些未经授权或非正常的用户行为。一旦发现诸如此类现象，可以采取各种措施，如封锁用户账号等。

9.4.1 利用who命令查询用户

为了查询当前系统中都有哪些用户，这些用户都来自哪里，可以使用who命令。who命令能够输出当前已注册到系统的所有用户，其语法格式如下：

```
who [-abdHlmpqrstTu] [file]
```

其中，“-a”选项表示同时选用其他多个选项，从而给出较多的信息。“-H”选项表示在输出数据之上增加一个标题行。“-r”选项用于查询系统当前所处的运行级。如果未指定文件名，表示读取并显示默认的/var/run/utmp文件中的内容，也即当前系统中的用户注册与活动信息。为了查询早期的历史记录，可以使用/var/log/wtmp文件。

who命令的输出数据通常包括下列字段：

```
NAME LINE TIME [IDLE] [PID] COMMENT [EXIT]
```

其中，NAME字段表示注册的用户名；LINE字段表示用户使用的终端设备名；TIME字段表示用户注册的起始时间；IDLE字段表示用户自上一个处理活动以来的空闲时间；PID字段表示用户的进程ID；COMMENT字段通常给出用户所在系统的主机名；EXIT字段表示用户的退出状态。

例如，下列信息表明当前系统中只有三个用户，三个用户均来自控制台终端（其中后两个用户是由于打开终端窗口注册的）：

```
$ who
gqxing  tty7      Nov 10 22:50 (:0)
gqxing  pts/0      Nov 10 01:38 (:0.0)
gqxing  pts/1      Nov 10 01:15 (:0.0)
$
```

如果使用“-H”选项，将会在输出数据之上增加一个标题行，以便了解每一行输出数据的意义，例如：

```
$ who -H
NAME      LINE      TIME          COMMENT
gqxing    tty7      Nov 17 15:45  (:0)
gqxing    pts/0     Nov 17 16:13  (:0.0)
gqxing    pts/2     Nov 17 16:41  (169.254.78.56)
$
```

使用“-a”选项可以显示更多的系统与用户活动数据，例如：

```
$ who -Ha
NAME      LINE      TIME          IDLE      PID  COMMENT  EXIT
          system boot  Nov 17 23:45
          run-level 2  Nov 17 23:45
LOGIN     tty5      Nov 17 23:45          812 id=5
LOGIN     tty4      Nov 17 23:45          806 id=4
LOGIN     tty2      Nov 17 23:45          821 id=2
LOGIN     tty3      Nov 17 23:45          822 id=3
LOGIN     tty6      Nov 17 23:45          824 id=6
LOGIN     tty1      Nov 17 23:45        1286 id=1
gqxing +  tty7      Nov 17 15:45      old      1487 (:0)
gqxing +  pts/0     Nov 17 16:13      old      2240 (:0.0)
gqxing -  pts/2     Nov 17 16:41      old      2738 (169.254.78.56)
$
```

9.4.2 利用finger命令查询用户

finger命令用于查询当前注册到系统的用户（包括本地和远程注册的用户），输出注册用户名及全名、所用终端的设备名、空闲时间、注册时间或远程主机名等。finger命令的语法格式简写如下：

```
finger [-lmsp] [user] [user@host]
```

finger命令的输出数据通常包含下列字段：

```
Login Name Tty Idle Login-Time Office Office-Phone
```

其中，Login字段表示注册的用户名；Name字段是用户的全名，也即passwd文件中的注释字段；Tty字段是用户注册时使用的终端设备名；Idle字段表示自上一个处理活动以来的空闲时间；Login-Time字段表示用户注册的日期和时间；Office字段包含用户所在系统的系统名或IP地址；Office-Phone字段给出的是用户的电话号码等信息。

例如，下列信息表明当前系统中只有三个用户，三个用户均来自控制台终端（其中后两个用户是由于打开终端窗口注册的）：

```
$ finger
Login      Name      Tty      Idle  Login Time   Office      Office Phone
gqxing    gqxing    tty7             Nov 17 15:45 (:0)
gqxing    gqxing    pts/0      15   Nov 17 16:13 (:0.0)
gqxing    gqxing    *pts/2             Nov 17 16:41 (169.254.78.56)
$
```

9.4.3 利用w命令查询用户活动

为了查询系统中现有的注册用户，以及每个用户当前正在做什么等信息，还可以使用w命令。w命令的语法格式如下：

```
w [-husfVo] [user]
```

其中，“-h”选项表示禁止输出包括标题的前两行信息。在输出的信息中，USER字段表示注册的用户名；TTY字段表示用户使用的终端设备名；FROM字段表示用户所在系统的名字；

“LOGIN@”字段表示用户注册的起始时间；IDLE字段表示空闲时间（从上一次输入命令后至今的持续时间）；JCPU字段表示从同一终端设备上运行的所有进程及其子进程使用的整个CPU时间；PCPU表示当前进程使用的CPU时间；WHAT字段表示当前进程的命令名及其参数。

如果未加任何选项，w命令的输出信息如下：

```
$ w
18:29:09 up 2:44, 3 users, load average: 0.09, 0.22, 0.17
USER      TTY      FROM    LOGIN@   IDLE   JCPU   PCPU   WHAT
gqxing    tty7     :0      15:45    ?      4:06   1.24s  gnome-session
gqxing    pts/0    :0.0    16:13    16:48   0.58s  0.58s  bash
gqxing    pts/2    bogon   16:41    0.00s   1.00s  0.02s  w
$
```

在上述命令输出信息中，第一行信息分别表示当前的系统时间、系统已经运行了多长时间、当前有多少个注册的用户，以及在过去的1分钟、5分钟和15分钟内系统的平均负载。

9.4.4 向注册用户发送消息

Linux系统提供大量的工具供注册的用户之间相互进行实时通信。通常，向注册用户发送日常消息的方法是利用/etc/motd（message of the day）文件。管理人员可以随时修改这个文件，一旦用户注册，即可见到motd文件的内容。

但上述方法仅在用户注册时有效，对于已经注册的用户，则看不到motd文件修改后的内容，除非再次注册。一旦有紧急情况需要通知用户，如因停电或系统故障需要立即停机时，可利用wall（write to all）命令向注册的用户发送紧急通知。例如：

```
$ wall
This system will shut down for emergency maintenance now.
You have 5 minutes to save your work and log out.
^D
```


不管用户当前正在做什么，wall命令将会把按下Ctrl-D组合键之前输入的文字信息立即写到用户的终端窗口中：

```
Broadcast Message from gqxing@iscas
(/dev/pts/0) at 18:24 ...

This system will shut down for emergency maintenance now.
You have 5 minutes to save your work and log out.
```

9.5 插件式认证模块

插件式认证模块（Linux Pluggable Authentication Modules, PAM）使系统管理员或应用程序能够决定如何利用通用的认证技术验证用户的身份。PAM提供一个共享模块库（位于/lib/security目录中），当需要认证用户时，应用程序只须调用共享模块库即可实现。位于/etc/pam.d目录中的每个配置文件均包含一组需要调用的模块或模块栈，这些模块决定了每个系统服务的认证处理方法。必要时，PAM也可以使用其他文件，如/etc/passwd，实现用户认证。插件式认证模块意味着用户能够随意增加或从认证栈中删除任何模块。

在应用程序开发过程中，程序员只须充分调用PAM提供的共享认证模块库，不必自己编写认证代码，即可实现用户认证要求。此外，对于给定的应用，系统管理员也能够利用PAM改变认证机制，而无须修改现有的应用。PAM为各种系统服务提供了认证机制，如login、ftp、su和sudo等。用户也可以利用认证栈技术的方法及其灵活性，采用不同的认证机制，组合一定的认证模块，集成一个诸如RSA、DCE或Kerberos的系统认证服务。

从用户注册到执行sudo命令，从执行su命令到关闭系统，无论何时需要用户提供密码验证，系统管理员都可以使用PAM配置认证处理过程，且对使用PAM实现认证的所有系统服务或应用程序而言，其处理过程基本上都是相同的。对于每一个系统服务（或应用程序），存储在/etc/pam.d目录中的配置文件描述了其认证处理过程。这些配置文件通常都有一个与系统服务（或应用程序）相同或类似的名字。例如，login注册认证的配置文件为/etc/pam.d/login。

在认证过程中，如果遇到问题，PAM发出的警告信息将会记录到/var/log/messages或/var/log/secure文件中，如果在修改PAM配置文件后没有达到预期的结果或出现异常，也可以查询这些日志文件，找出问题所在。为了防止图谋不轨的用户看到PAM信息，PAM仅把出错信息写入日志文件而非送到终端屏幕。

9.5.1 配置文件、模块类型与控制标志

PAM采用/etc/pam.conf或/etc/pam.d目录中以各种服务命名的文件作为配置文件，配置文件中的每一行表示PAM需要在认证过程中执行的处理动作。如果/etc/pam.d目录存在，PAM将会忽略/etc/pam.conf配置文件。在/etc/pam.d目录中，每个配置文件的语法格式如下：

```
type control module-path [module-arguments]
```

其中，type是认证模块类型标识符，表示当前模块属于哪一种模块类型；control是如何处理认证的标志；module-path是PAM模块文件的完整路径名，也可以是针对默认存储位置（/lib/security）的相对路径名；作为认证模块的参数，module-arguments是选用的。如果存在，可以是一个参数或多个由空格分隔的参数表，用于调整给定模块的处理功能。

认证模块类型用于区分不同性质或适用不同处理需求的各种认证模块。当前PAM主要支持account、auth、password及session四种不同类型的认证模块。表9-9给出了各种认证模块的说明。

表9-9 认证模块类型

模块类型	控制功能描述
account	用于实现认证之外的用户账号管理。此类模块通常主要用于限制或允许访问基于时间、当前可用系统资源（如最大用户数）以及用户所在位置（如超级用户root只能在系统控制台注册）的服务。例如，检测用户在当前的时间点是否能够访问相应的服务
auth	采用用户名、密码及其他方式认证用户，授权用户使用相应的服务。此类模块主要提供两种用户认证模式：首先，提示用户输入用户名、密码或其他标识信息，获知用户的身份。其次，根据用户提供的身份信息，赋予用户一定的用户组成员权限或其他权限
password	修改用户密码。此类模块主要用于提请用户更新密码等信息。对于每一种基于“提示/响应”的认证方式，通常都存在一个相应的认证模块，以实现相应的处理过程。实际上，password类型的认证模块并不常用
session	此类模块通常主要用于会话启停管理。在服务开始运行（如用户注册）时，事先设置相应的运行环境，如记录日志，安装用户主目录等；当服务终止运行（如用户注销）时，执行相应的善后处理过程，如关闭日志、卸载用户主目录等

account认证模块的功能类似于auth认证模块，其主要差别是仅当用户通过认证之后才能调用前者。account认证模块可以作为一种附加的安全检测措施，以便在用户能够访问系统之前再做进一步的限定检测。例如，可以利用account认证模块强制用户只能在上班时间注册到系统等。

session认证模块用于启动和终止一个用户会话。在Linux系统中，一个通用的会话模块是pam_mail，当用户注册到传统字符界面的系统环境时，可以使用这个模块通知用户是否有新的邮件到来。

认证控制标志用于控制怎样进行认证，以及如何处理认证模块返回的认证结果。在PAM配置文件中，可以采用表9-10所示的关键字，设置认证模块的控制标志。

表9-10 认证控制标志

控制标志	功能说明
required	要求在指定的认证模块都运行成功之后才能报告整个认证成功。也就是说，认证栈中的所有模块都必须执行一次，PAM才能报告整个认证究竟是成功还是失败。这种认证技术用于延迟程序调用的返回结果，直至所有的认证模块都执行完毕，从而防止黑客确切地了解究竟在哪个认证环节出现了问题，增加系统入侵的难度
requisite	类似于required，指定的认证模块都必须运行成功才能保证认证栈认证成功。如果存在多个认证模块，一个模块的认证成功并不能保证整个认证栈认证成功。如果某个模块的认证失败，PAM将会停止执行后续的认证，立即返回调用程序，报告认证失败。这种认证技术相当于多项认证技术的逻辑与运算，只有所有的模块都认证成功才能满足要求。在实际运用中，这种认证方法能够防止用户在一种不安全的网络连接中仅靠输入正确的密码访问系统
sufficient	如果当前模块的认证已经成功，表示整个认证栈认证成功，无须再执行随后的其他认证模块。如果认证失败，也并不影响整个认证栈的最终认证结果，PAM将会采用后续模块继续执行认证。这种认证技术相当于多项认证技术的逻辑或运算，只要认证栈中的任何一种认证能够成功就算满足要求。例如，当使用rsh连接到远程系统时，pam_rhosts_auth模块首先会检测当

(续表)

控制标志	功能说明
	前的连接是否为可信任的。如果连接是可信任的, pam_rhosts_auth模块将会报告认证成功, 继而PAM也会立即报告rsh守护进程认证成功。此时不会提示用户输入密码。如果连接不是可信任的, PAM再开始执行其他认证模块, 如提示用户输入密码。如果后者认证成功, PAM将会忽略pam_rhosts_auth模块报告的认证失败结果。如果两种认证都不成功, 用户注册将会遭到拒绝
optional	认证结果通常可以忽略。仅当服务的整个认证栈中只有一个认证模块时, 选用模块的认证结果才有效
include	插入指定的配置文件, 逐行引用其中的全部内容

PAM可以使用应用程序要求的各类认证模块, 实现用户账号状态检测、会话管理及密码修改等认证方式。PAM可用的认证模块通常均位于/lib/security目录中。

/etc/pam.d目录中的各种配置文件定义了执行用户认证时需要用到的一组认证模块。同一类型的认证模块称做认证栈, PAM将会按照配置文件中的列举顺序, 从栈顶开始依次调用每一个认证模块。每个认证模块依次向PAM返回认证成功或失败的结果, 根据认证模块的控制标志, 再由PAM向调用程序返回整个处理过程的认证结果。



注意 PAM认证的成功与失败只是一种说明性的概念, 与Linux系统中的真假概念是完全不同的。

除了采用required、requisite、sufficient及optional等关键字表示控制标志之外, PAM还支持采用下列复杂的语法格式表示的控制标志:

```
[value1=action1 value2=action2 ...]
```

其中, valueN相当于认证模块的返回值, 如success、ignore、new_authtok_reqd、try_again、abort、bad_item、incomplete和default等。default意味着没有明显定义的其他返回值。

actionN可以是一个整数n(表示下一步处理动作是跳转到认证栈中的第n个模块), 也可以是ignore、bad、die、ok、done和reset等。其中, ignore表示当前模块的返回状态不影响整个认证栈的最终认证结果, 可以忽略不计。bad表示当前模块的认证失败。如果当前模块是认证栈中的第一个模块, 其认证结果可以看做整个认证栈的认证结果。die等同于bad, 不同之处是立即终止整个认证栈, 返回认证失败。ok表示当前模块的认证可以看做整个认证栈的最终认证结果。注意, 如果认证栈中先前模块的认证结果表示失败, 这个认证结果不能取代先前的认证结果。done相当于ok, 不同之处是立即终止认证栈, 返回认证成功。reset表示清除认证栈先前所有模块的认证结果, 从下一个模块开始继续认证。

如果采用第二种语法格式, 前述的四个关键字required、requisite、sufficient和optional可分别表示如下:

- required [success=ok new_authtok_reqd=ok ignore=ignore default=bad]
- requisite [success=ok new_authtok_reqd=ok ignore=ignore default=die]
- sufficient [success=done new_authtok_reqd=done default=ignore]
- optional [success=ok new_authtok_reqd=ok default=ignore]

下面的login文件是PAM配置文件的一个实例（采用grep命令的目的是剔除以注释符“#”为起始字符的大量注释行），主要用于处理用户的系统注册过程：

```
$ grep '^[^#]' /etc/pam.d/login
auth      optional    pam_faildelay.so  delay=3000000
auth      requisite   pam_securetty.so
auth      requisite   pam_nologin.so
session   [success=ok ignore=ignore module_unknown=ignore default=bad] pam_
selinux.so close
session   required    pam_env.so readenv=1
session   required    pam_env.so readenv=1 envfile=/etc/default/locale
@include  common-auth
auth      optional    pam_group.so
session   required    pam_limits.so
session   optional    pam_lastlog.so
session   optional    pam_motd.so
session   optional    pam_mail.so standard
@include  common-account
@include  common-session
@include  common-password
session   [success=ok ignore=ignore module_unknown=ignore default=bad] pam_
selinux.so open
$
```

在上述配置文件中，每一行的第一个字段是模块类型标识符，如auth及session等。第二个字段是控制标志，表示一旦认证失败时PAM应采取的处理动作。从第三个字段开始是PAM模块文件（位于/lib/security目录）的相对路径名或选用的参数。PAM模块库利用/etc/pam.d目录中的配置文件确定由哪一些模块负责相应的认证处理。以“@include”为起始字符串的文本行表示在相应的位置插入指定的文件。

现以上述的/etc/pam.d/login配置文件为例，说明用户注册的认证过程。我们知道，注册过程是由login程序引导用户输入用户名和密码实现用户认证的。login命令首先提示用户输入一个用户名，然后调用PAM，依次启动一系列认证模块，开始用户的认证过程。

（1）按照login文件列举的顺序，PAM首先调用pam_faildelay模块，设置一旦认证失败之后，系统允许用户再次尝试注册的延迟时间。“delay=3000000”表示延迟时间为3000000微秒。

（2）pam_securetty模块用于确保超级用户只能从指定的终端（如控制台）注册。若想保证整个认证结果的成功，必须确保pam_securetty模块的认证结果是成功的。在这一认证过程中，仅当有人试图从禁用的终端中以超级用户的身份注册，pam_securetty模块才会报告失败。如果接受认证的用户并非超级用户，或者即使是超级用户，但用户是从允许使用的终端注册的，pam_securetty模块仍会报告认证成功。

（3）采用pam_nologin认证模块时，如果/etc/nologin文件存在，系统只允许超级用户root注册，而禁止任何普通用户注册。一旦普通用户尝试注册，系统将会显示nologin文件的内容，说明禁止注册的原因。也就是说，仅当nologin文件不存在，或当前注册的用户是root，pam_nologin模块才能报告认证成功。

（4）pam_selinux模块用于设置安全环境。

（5）pam_env模块用于设置环境变量。除非使用envfile和readenv参数指定了其他文件，默

认的变量设置取自/etc/security/pam_env.conf配置文件,同时也利用/etc/environment文件设置环境变量。“readenv=1”是一个控制开关,表示是否允许(0表示禁止,1表示允许)使用envfile参数指定的文件设置环境变量。“envfile=/etc/default/locale”表示使用/etc/default/locale文件替换默认的变量设置。当不同的服务需要不同的变量设置时,envfile参数是非常有用的。

(6) @include引入的common-auth文件含有专门检测注册用户是否合法的认证模块,可作为认证栈的补充,用于验证用户名与密码。

(7) pam_group模块实际上不进行用户认证,它只是根据用户申请的服务,向用户授予用户组成员资格。默认的用户组成员资格取自/etc/security/group.conf配置文件。pam_group模块可与/etc/group文件并存,如果pam_group模块授予用户任何用户组资格,这些用户组可以作为/etc/group文件的补充。

(8) pam_limits模块用于设置用户在会话期间的系统资源限制。通常,默认的系统资源限制取自/etc/security/limits.conf配置文件。/etc/security/limits.d目录中的文件可以用做进一步的修正。

(9) pam_lastlog模块用于显示用户上一次注册的时间,也用于维护/var/log/lastlog日志文件。

(10) 在成功注册之后,可以使用pam_motd模块显示/etc/motd文件中的通告信息。

(11) pam_mail模块用于提供邮件检测服务。如果发现用户有新到的邮件,输出“You have new mail”提示信息。

9.5.2 修改PAM配置文件

有些UNIX或Linux系统要求用户必须是超级用户组(俗称wheel组,实际上就是root用户组)中的一个成员,才能使用su命令。尽管Ubuntu Linux系统并未采用这种配置方法,但通过编辑/etc/pam.d/su文件,仍可利用PAM达到同样的目的

```
$ cat /etc/pam.d/su
.....
# Uncomment this to force users to be a member of group root
# before they can use `su'. You can also add "group=foo"
# to the end of this line if you want to use a group other
# than the default "root" (but this may have side effect of
# denying "root" user, unless she's a member of "foo" or explicitly
# permitted earlier by e.g. "sufficient pam_rootok.so").
# (Replaces the `SU_WHEEL_ONLY' option from login.defs)
# auth      required    pam_wheel.so

# Uncomment this if you want wheel members to be able to
# su without a password.
# auth      sufficient  pam_wheel.so trust
.....
$
```

上述“# auth required pam_wheel.so”一行前面存在一个注释符“#”。如果删除了注释符,意味着只有root用户组的成员才能使用su命令(其认证控制标志为required)。此外,如果删除“auth sufficient pam_wheel.so trust”一行最前面的注释符“#”,表示允许root用户组的成员在运行su命令时不必提供密码(其认证控制标志为sufficient)。



除非确实了解怎样配置PAM，一般不要轻易修改/etc/pam.d目录中的文件。修改PAM配置文件的错误，有可能导致用户无法正常访问系统。为避免此类问题，在修改之前应事先备份PAM配置文件，同时打开一个新的终端窗口，使用“sudo -i”命令进入一个超级用户Shell环境。然后再仔细地测试修改后的配置文件，确保能够正常注册，新的设置能够达到预期的目的。如果出现错误，可以在打开的超级用户终端窗口，利用备份的配置文件恢复修改有误的文件。

9.6 超级用户与sudo命令

超级用户root拥有无限的权利，在长时间的系统访问过程中，如果运用不当，或在无意中出現失误，有可能损坏系统，如破坏文件系统等，严重时甚至可能引起系统瘫痪。有些系统特权命令还可能会侵犯用户的隐私，或危及系统的安全。为了确保Linux系统能够安全地正常运行，Ubuntu Linux系统不允许使用root直接注册。当需要以超级用户root的身份和权限运行特权命令，执行系统管理与维护任务时，Ubuntu Linux系统至少提供下列3种途径或方法，使用户能够获得超级用户的访问权限：

- 利用sudo命令，可以临时获取超级用户的访问权限，执行系统管理与维护等特权命令，运行结束之后，立即恢复用户本来的身份及访问权限。但是，Ubuntu Linux系统只允许admin用户组的成员运行sudo命令。
- 利用su或“sudo -i”命令进入超级用户的Shell运行环境，可以在较长的时间内，以超级用户的身份和权限访问系统，直至运行exit命令或按Ctrl-D组合键退出。但在使用su命令之前，必须首先解除超级用户账号的封锁（即为root用户账号设置密码）。在Ubuntu Linux系统中，sudo命令完全可以替代su命令。
- 启用超级用户root的账号，使用root直接注册系统，在整个会话过程中，一直以超级用户的身份和权限访问系统，直至退出系统。

9.6.1 超级用户的访问控制

利用/etc/securetty和/etc/security/access.conf等配置文件，PAM能够控制或限定哪些用户，何时以及如何以超级用户root的身份注册到系统。例如，/etc/securetty文件用于控制用户能够从哪些网络或终端上，以超级用户root的身份注册，/etc/security/access.conf文件又进一步增加了用户与注册方式的组合控制措施。

禁止用户以超级用户root的身份从网上注册是Ubuntu Linux系统采取的一项默认的安全策略，这一安全措施就是利用PAM安全模块与/etc/security/access.conf文件实现的。access.conf配置文件包含允许以超级用户root身份注册的所有用户以及网络与终端访问方式。在初始安装Ubuntu Linux系统之后，这个文件中均为注释行，表示禁止任何用户在网络上以超级用户root的身份注册访问。

/etc/security/access.conf配置文件可以看做一个注册访问控制表，用于指定系统接受或拒绝注册的用户及其注册方式（如通过TTY设备、主机或网络等方式访问系统）。当用户注册到系统时，系统将会扫描这个文件，找出第一个匹配的“用户/TTY设备”、“用户/主机”或“用户/网络”组合，根据其中的访问权限定义，可以确定究竟是接受还是拒绝用户的注册。access

.conf文件的每一行由三个字段组成, 中间以冒号“:”作为分隔符, 其语法格式如下:

```
permission:users:origins
```

其中, permission字段可以是一个加号“+”(表示允许相应的用户注册)或减号“-”(表示拒绝相应的用户注册)。users字段是一个或多个注册的用户名、用户组名或ALL(表示可以匹配任何用户或用户组)。origins字段可以是一个或多个TTY设备名(表示非网络连接的注册)、主机名、域名(最前面需增加一个句点“.”前缀)、主机IP地址、网络号(最后面需附加一个句点“.”后缀)、网络地址加子网掩码(子网掩码可以是一个十进制的数值或IP地址)、ALL(表示匹配任何设备)或LOCAL(表示匹配不包含句点“.”的任何字符串)。

例如, 下面的例子表示允许超级用户root通过X终端“:0”与pts/0~pts/6等终端设备注册访问系统:

```
+ : root : :0 pts/0 pts/1 pts/2 pts/3 pts/4 pts/5 pts/6
```

下面的例子表示用户能够从指定IP地址(192.168.100.1、192.168.100.2和192.168.100.6)的主机, 以及本地系统(127.0.0.1)中, 以超级用户root的身份注册访问系统:

```
+ : root : 192.168.100.1 192.168.100.2 192.168.100.6
+ : root : 127.0.0.1
```

下面的例子表示, 除了采用sudo或su命令获取超级用户访问权限的途径之外, 禁止超级用户root以任何方式直接注册访问系统:

```
- : root : ALL
```



利用OpenSSH, 用户仍然能够以超级用户root的身份从网上注册。这是因为在Ubuntu Linux系统中, ssh并不受securetty安全模块或access.conf文件的限制。此外, 在Ubuntu Linux系统的/etc/ssh/sshd_config文件中, PermitRootLogin已设置为yes, 这意味着允许使用ssh以超级用户root的身份从网上注册。

9.6.2 利用sudo运行特权命令

传统上, 用户大都喜欢采用超级用户root直接注册到系统, 或使用su命令获取超级用户的访问权限。出于系统安全方面的考虑, Ubuntu Linux系统强烈建议用户使用sudo命令运行特权命令。事实上, 在安装Ubuntu Linux系统之后, 尽管系统也提供了root用户账号, 但由于未设置密码, 从而锁住了超级用户账号, 无法用以注册, 也无法利用su命令等传统方法成为超级用户(系统启动后直接进入恢复模式时除外)。

为了使部分用户也有机会运行特权命令, 处理系统管理与维护任务, Ubuntu Linux系统提供了sudo命令及其相关机制, 以替代传统的超级用户权限获取方法。利用sudo命令, 能够像利用root注册成为超级用户一样, 在命令行界面中运行任何系统管理方面的特权命令。sudo命令的语法格式简写如下:

```
sudo [-b] [-u user] command
sudo [-ikLls]
```

其中, “-b”选项表示以后台方式运行指定的命令; “-u”选项表示使用指定用户的身份和权限执行指定的命令, 如果未加此选项, sudo将会以超级用户的身份和权限运行当前的命令;

command 可以是准备执行的任何命令或Shell脚本。“-i”选项表示调用超级用户root的注册Shell，运行其初始化文件（如profile），在超级用户的Shell环境中访问系统；“-k”选项表示重新设置sudo命令的时戳，这意味着，当再次运行sudo命令时，必须重新输入sudo密码；“-L”选项用于显示当前用户在/etc/sudoers文件Defaults一行中能够设置的参数；“-l”选项用于显示当前用户在本地系统中能够利用sudo运行的命令；“-s”选项表示调用超级用户的注册Shell（类似于“-i”选项，但不改变现有的运行环境）。

当需要以超级用户的身份和权限运行特权命令，执行系统管理与维护任务时，可在命令行界面中使用sudo命令，即在实际运行的命令前面增加一个sudo前缀，当提示用户输入密码时，输入sudo密码。密码通常会保存15分钟，在此期间，如果再次运行sudo命令，则无需提供密码。超过此限，需要重新输入密码。sudo命令将会影响同一命令行上的所有命令，直至遇到换行符或另一个sudo命令。

但是，sudo命令不能调用图形界面的程序。如果想在命令行中运行图形界面的系统管理程序，Ubuntu Linux系统提供了一个gksudo命令（gksudo又调用sudo命令实现用户认证），其使用方式与sudo命令完全相同（在KDE桌面环境中，可以利用kdesu命令启动图形界面的管理程序）。



并非任何用户都能运行sudo命令，按照系统的默认配置，只有admin用户组的成员才能利用sudo命令运行特权命令。

当第一次运行sudo命令时，sudo通常会提示用户输入密码（注意，此时不要输入超级用户root的密码），如果输入的sudo密码是正确的，sudo命令将会开始计时，如果在15分钟之内继续运行sudo命令，sudo不会再提示用户输入密码。

下面的例子表示，普通用户gqxing无权直接使用date命令设置系统时钟。但由于gqxing也是admin用户组的成员，故可运行sudo命令。当利用sudo运行date命令设置系统时钟时，sudo命令在收到正确的密码后，便能确保用户成功地执行date命令：

```
$ date 122218502009
date: cannot set date: Operation not permitted
Tue Dec 22 18:50:00 CST 2009
$ sudo date 122218502009
[sudo] password for gqxing:
Tue Dec 22 18:50:00 CST 2009
$
```

使用“-l”选项运行sudo命令时，可以查询当前用户能够利用sudo命令运行哪些命令。由于gqxing是安装系统时创建的用户，因而也是admin用户组中的一个成员，故能够以任何用户（包括超级用户）的身份运行任何命令（包括特权命令）：

```
$ sudo -l
Matching Defaults entries for gqxing on this host:
env_reset

User gqxing may run the following commands on this host:
(ALL) ALL
$
```


当有若干特权命令需要以超级用户的身份和权限运行时，最简单的方法是启动一个超级用户的Shell运行环境，直至运行完所有的特权命令之后，再利用exit命令或按下Ctrl-D组合键，退出超级用户的Shell环境，而无需在每次执行特权命令时都输入一个sudo命令前缀。为此，可以使用带有“-i”选项的sudo命令：

```
$ sudo -i
[sudo] password for gqxing:
# id
uid=0(root) gid=0(root) groups=0(root)
# pwd
/root
# exit
logout
$
```

在上述情况下，调用“sudo -i”命令将会进入超级用户的运行环境。如果仅想拥有超级用户的访问权限，而保持用户当前的运行环境不变，可以使用“sudo -s”命令：

```
$ sudo -s
# id
uid=0(root) gid=0(root) groups=0(root)
# pwd
/home/gqxing
# exit
exit
$
```

9.6.3 sudoers配置文件

在Ubuntu Linux系统中，sudo命令需根据/etc/sudoers配置文件的设置，确定是否赋予用户访问特权命令的权限。sudoers配置文件的初始设置不一定完全能够适应用户的实际需求，有时可能需要调整其配置。为了修改sudoers文件，最好的方法是采用专用的编辑程序visudo：

```
$ sudo visudo
```

visudo编辑程序能够提供加锁、编辑以及语法检测等功能。通常，visudo主要调用nano编辑器修改sudoers文件。通过设置VISUAL环境变量，用户也可以使用自己熟悉的编辑器，如vi/vim，编辑sudoers文件：

```
$ export VISUAL=vi
```



如果直接使用vi/vim等文本编辑器修改/etc/sudoers文件，当出现语法错误时，可能会导致用户无法利用sudo命令获得超级用户的访问权限。因此，强烈建议用户使用visudo命令修改sudoers文件。visudo程序能够检测语法错误。一旦出现错误，visudo不会允许用户轻易写入sudoers文件，且提供一个修正错误的机会：或者重新编辑，或者放弃修改而退出，最后一个选择是强制保存修改（这是一个比较危险的选择，visudo标记为DANGER!）。

/etc/sudoers文件含有两种类型的信息：用户别名与用户权限规范的定义，每个定义占用一行。通过增加反斜杠“\”后缀，也可以延续到后续行。此外，以注释符“#”为起始字符的行是注释行，注释行可以出现在任何位置。

1. 用户权限规范

在/etc/sudoers文件中，定义用户权限规范的语法格式如下（等号“=”前后的空格可有可无）：

```
user_list host_list = [(runas_list)] [tag_list] command_list
```

在用户权限规范的语法格式中，每个字段的意义及用途说明如下：

- **user_list**——用于指定单个或一组用户，表示当前定义的权限适用的用户对象。指定用户时可以采用用户名、用户组（增加一个百分号“%”前缀）和用户别名。此外，还可以使用内部定义的别名ALL，表示所有的用户，意味着当前的权限定义适用于所有的用户。
- **host_list**——用于指定当前的权限定义施加的主机对象。主机对象可以是一个或多个主机名、IP地址或主机别名。此外，还可以使用内部定义的别名ALL，表示所有的主机系统，意味着当前的权限定义适用于含有此sudoers文件的所有系统。
- **runas_list**——类似于user_list，用于指定单个或一组用户，两者唯一的差别是指定runas_list时可以采用用户ID。当使用“-u”选项调用sudo命令时，command_list中定义的命令能够以这里指定的用户身份运行（同样，指定的用户可以是用户名、加百分号“%”前缀的用户组名和用户别名）。runas_list是选用的，如果省略了runas_list，意味着sudo以超级用户root的身份运行command_list中指定的命令。
- **tag_list**——用于设置或限定某些特殊的命令。例如，可以使用PASSWD和NOPASSWD限定用户执行某些管理和维护命令时是否需要提供密码，使用NOEXEC能够防止任何程序启动新的Shell（因为一旦使用sudo命令运行某个程序，程序在整个运行期间都会拥有超级用户的访问权限，因而也能够启动拥有超级用户特权的Shell，最终绕过sudoers文件设定的任何限制）。
- **command_list**——指定权限定义适用的实用程序。指定实用程序时可以采用程序的名字、存有实用程序的目录名和命令别名，而且所有的文件名或目录名必须是绝对路径名。如果命令后面附加一个空的双引号，表示用户不能在运行时指定任何命令行参数和选项。此外，也可以在命令后面指定固定的参数或通配符，限制用户能够使用的参数。

下面是系统提供的sudoers文件，其中的“%admin ALL=(ALL) ALL”表示用户组admin中的任何成员均可利用sudo命令获取超级用户的访问权限，运行系统管理与维护方面的特权命令：

```
$ sudo cat /etc/sudoers
.....
Defaults          env_reset
.....
# User privilege specification
root    ALL=(ALL) ALL

# Uncomment to allow members of group sudo to not need a password
# (Note that later entries override this, so you might need to move
# it further down)
# %sudo ALL=NOPASSWD: ALL
```

```
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
$
```

下列用户权限规范使用户cathy能够在含有此sudoers文件的所有系统（ALL）中使用sudo运行mount和umount命令，安装和卸载文件系统：

```
cathy ALL=(root) /bin/mount, /bin/umount
```

runas_list是选用的，如果省略了runas_list，sudo允许普通用户以超级用户root的身份运行用户权限规范中定义的命令。在下面的例子中，cathy用户可以利用sudo命令，在命令行中使用umount命令卸载/paper文件系统：

```
$ umount /paper
umount: only root can unmount /dev/sdb7 from /paper
$ sudo umount /paper
[sudo] password for cathy:
$
```

如果把上述的sudoers文件改写如下，则意味着不允许用户hwang卸载/paper文件系统，但可以使用mount和umount命令安装或卸载其他任何文件系统：

```
hwang ALL=(root) /bin/mount, /bin/umount, !/bin/umount /paper
```

在此情况下，当用户hwang使用sudo运行下列mount命令时，将会遭到拒绝：

```
$ sudo umount /paper
Sorry, user hwang is not allowed to execute '/bin/umount /paper' as root on localhost.
$
```

2. 别名

别名机制用于定义或组合一组用户、主机或命令等，以便在sudoers文件中引用，其语法格式如下：

```
alias_type alias_name = alias_list
```

其中，alias_type用于定义不同类型的别名，如User_Alias、Runas_Alias、Host_Alias以及Cmnd_Alias等；alias_name是一个便于引用的别名，按惯例，别名通常均采用大写字母命名；alias_list是由一个或多个名字（如用户名、主机名和命令等）组成的名字列表，列举的名字之间需加空格或逗号“,”分隔符。别名定义中也可以嵌套其他别名，例如，在指定用户时可以使用先前定义的用户别名。系统提供一个内部定义的别名ALL，可以匹配其所在位置的任何对象，包括用户名或系统等。例如，如果在用户名位置使用ALL，意味着匹配所有的用户。如果某个名字前面加有感叹号“!”前缀，意味着取其反义。

- User_Alias——用于定义用户别名，以便引用一组限定的用户。指定的用户可以是单个用户名或用户组（用户组名字前面需加一个百分号“%”前缀，如%admin，表示用户组admin的所有成员）等。下面的例子表示sudoers文件定义了三个用户别名：OFFICE、ADMIN和ADMIN2。用户别名OFFICE包括cathy、hwang和wzhang三个用户；用户别名ADMIN包括wzeng、qchen以及admin用户组的所有成员；用户别名ADMIN2包括除dzhao之外的所有admin用户组成员。

```
User_Alias OFFICE = cathy, hwang, wzhang
User_Alias ADMIN = wzeng, qchen, %admin
User_Alias ADMIN2 = %admin, !dzhao
```

- Runas_Alias——也用于定义用户别名，以便指定的用户能够以超级用户之外的其他用户身份运行指定的命令。下面定义的用户别名GUEST表示用户能够以guest或visitor的用户身份运行指定的命令：

```
Runas_Alias GUEST = guest, visitor
```

- Host_Alias——用于定义主机别名，以便引用一组限定的主机。仅当在多个主机上运行的sudo命令引用的是同一sudoers文件时，主机别名才有意义。如果想要采用规范域名，如iscas.abcnet，而非iscas，必须设置fqdn标志。注意，在定义主机别名时采用规范域名会影响sudo命令的运行效率。下面定义的主机别名TESTING包括iscas与sinosoft两个主机系统：

```
Host_Alias TESTING = iscas, sinosoft
```

- Cmnd_Alias——用于设置命令别名，以便引用一组限定的命令。下列命令别名定义了一个命令文件和一个目录（其后面附有一个斜杠字符“/”，表示目录中的所有命令文件）：

```
Cmnd_Alias BASIC = /bin/cat, /usr/bin/vim, /bin/df, /usr/local/bin/
```

下面的例子说明了怎样使用别名。在使用sudo命令运行shutdown、halt或poweroff等命令关闭系统时，通常需要提供sudo密码。为了避免总是输入密码，可以定义下列命令别名：

```
Cmnd_Alias SHUTDOWN_CMDS = /sbin/shutdown, /sbin/halt, /sbin/poweroff
```

然后，在sudoers配置文件中“%admin ALL = (ALL) ALL”一行的后面再额外增加下列权限规范定义即可：

```
%admin ALL=(ALL) NOPASSWD: SHUTDOWN_CMDS
```

此外，还可以彻底禁止运行sudo命令时提示用户输入密码。如前所述，第一次使用sudo命令时，用户必须提供密码，才能运行随后的系统管理与维护命令。密码的有效持续时间是15分钟，如果超过此限，再次使用sudo命令时仍须提供密码。为了继续使用sudo命令获取超级用户的访问权限，但禁止sudo命令提示用户输入密码，可以编辑sudoers文件，在文件的后面增加下列配置：

```
user_or_group_name ALL=NOPASSWD: ALL
```

例如，下述设置表示，用户gqxing任何时间运行sudo命令时，均不需要提供密码：

```
gqxing ALL=NOPASSWD: ALL
```

3. 默认值

利用关键字Defaults，可以修改配置选项的默认值。在sudoers文件中，大多数配置选项的值都是布尔值on或off，部分配置选项的值可以是一个字符串。在一个Defaults定义行中，增加一个配置选项表示启用相应的标志，如果在配置选项前增加一个感叹号“!”前缀则意味着关闭相应的标志。例如，sudoers初始文件中的下列Defaults定义行表示启用env_reset标志：

```
Defaults env_reset
```

下面是部分常用的配置选项及其默认值（完整地说明可以查阅sudoers手册）。

- `fqdn=on/off`: 如果想在sudoers文件中使用规范域名, 应当启用这个标志, 以便能够调用DNS服务器实现主机域名解析。在此情况下, 可以混合使用简单的主机名或规范域名。采用规范域名的缺点是影响sudo运行的性能, 尤其是当DNS系统工作不正常时。一旦设置这个标志, 必须引用主机的实际域名, 而不能使用CNAME记录定义的别名。注意, 如果hostname命令返回的是规范域名, 不需要设置这个标志。这个标志的默认值是on。
- `passwd_tries=num`: 设置允许用户尝试输入密码的次数。当用户输入的密码不正确时, sudo允许用户尝试输入其他密码, 超过此限, sudo不会再提示用户, 并立即退出。默认值是3。
- `rootpw=on/off`: 设置这个标志之后, 当调用的sudo命令提示用户输入密码时, 只能输入超级用户root的密码。由于sudo命令输出的提示信息完全相同, 启用这个标志后可能会给用户造成混淆。此外, 如果超级用户root没有解锁（即未设置密码）, 不要启用这个标志, 否则无法运行sudo命令。一旦出现此类问题, 可以把系统引导至恢复模式, 然后关闭这个标志。这个标志的默认值为off, 意味着sudo提示的是安装系统时创建的第一个用户的密码。如果rootpw设置为on, 且已经为超级用户root设置了密码, 当调用要求提供密码的图形界面程序时, 也需要提供超级用户root的密码。
- `shell_noargs=on/off`: 如果设置了这个标志, 当未加任何参数调用sudo命令时, 系统将会调用超级用户root的Shell, 但不改变当前的运行环境, 其效果等同于使用“-s”选项调用sudo命令。这个标志的默认值是off。
- `timestamp_timeout=mins`: 用于设置sudo命令密码保持有效的最小时间长度（分钟数）。这个时效的默认值是15。如果把把这个值设置为-1, 表示密码的时效没有限制。

9.6.4 admin用户组成员的访问权限

在初始安装Ubuntu Linux系统之后, sudoers文件包含下列用户权限规范:

```
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

在上述用户权限规范的定义中, 等号“=”左边的ALL表示这个权限规范适用于所有的主机系统。正如其注释行所言, %admin一行定义的用户权限规范表示admin用户组的所有成员都能够以任何用户（中间的ALL）的身份运行任何命令（最后面的一个ALL）, 尤其能够利用sudo命令, 以超级用户的身份和权限运行任何特权命令。

此外, 如果使用“(root)”替代“(ALL)”或省略“(ALL)”, 用户仍然能够使用超级用户的权限运行任何命令, 但不能使用“-u”选项以其他用户身份运行命令（如果拥有超级用户的访问权限, 这个限制通常不会造成问题）。

安装系统时创建的第一个用户, 除了同名的用户组, 也会自动加到admin用户组中, 因而具有运行系统特权命令, 执行系统管理任务的访问权限。

9.6.5 直接使用root注册

在初始安装的Ubuntu Linux系统中, 超级用户root的账号没有设定密码, 因而不能使用root

直接注册。如果想要运行系统特权命令，唯一的方法就是在实际运行的命令之前增加sudo前缀。如果想在注册界面中直接使用超级用户root注册，首先需要启用超级用户账号，即为root用户账号设置密码。为此，可在终端窗口中输入下列sudo和passwd命令，根据提示首先输入sudo密码，然后设置超级用户的密码：

```
$ sudo passwd root
[sudo] password for gqxing:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
$
```

其次，使用sudo命令和任何文本编辑器，修改/etc/gdm/gdm.conf配置文件，把其中的“AllowRoot=false”一行改为“AllowRoot=true”。

此外也可以在GNOME桌面环境中，选择“系统→系统管理”菜单，在“认证”对话框中输入sudo密码，在“登录窗口首选项”界面中单击“安全”标签，然后选中“允许本地系统管理员登录”复选框。

此后，再次注册时就能够以超级用户root直接注册了。之后，如果需要再次封锁超级用户root的账号，可以使用下列命令：

```
$ sudo passwd -l root
$
```

9.6.6 以其他用户身份访问系统

总是使用sudo运行特权命令也并非最佳选择。在一个多用户的共享系统中，尤其是在一个集中管理的服务器中，使用sudo命令可能比使用su命令更安全一些。但在一个单用户的个人系统中，经常使用sudo命令与审慎地使用su命令并没有太大的差别。作为一个UNIX/Linux系统的资深用户，如果不习惯或根本就不喜欢使用sudo命令，最简单易行的方法就是解除超级用户账号的封锁，以便随时能够使用su命令，获取超级用户的访问权限，甚至进入超级用户的Shell运行环境。

su (substitute user) 命令的主要功能是改变用户的身份，使普通用户能够成为超级用户，当然也可以变身为其他普通用户。利用su命令，能够以指定的用户或超级用户root（默认值）的身份和权限，调用新的注册Shell，在新的运行环境中访问系统。通过运行exit命令或按下Ctrl-D组合键，可以返回先前的Shell工作环境。当然，这样做的前提是需要知道超级用户root或其他用户的密码。

单独运行su命令而不加任何选项与参数时，系统将会调用具有超级用户权限的Shell，进入超级用户的Shell工作环境，改变当前用户的有效用户ID与用户组ID，但会保持先前的工作目录不变。下面的例子说明了用户cathy使用su命令前后的环境变化情况：

```
$ id
uid=1001(cathy) gid=1001(cathy) groups=1001(cathy)
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
$ pwd
```

```

/home/cathy
$ su
Password:
# id
uid=0(root) gid=0(root) groups=0(root)
# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
# pwd
/home/cathy
#

```

当以“su -”、“su -l”或“su -login”形式运行su命令时，可以启用超级用户的Shell工作环境，直接进入超级用户的主目录，如同使用root注册一样。不仅是用户ID与用户组ID，整个运行环境也完全等同于超级用户root的运行环境。下面的例子说明了用户hwang使用“su -”命令前后的环境变化情况：

```

$ id
uid=1002(hwang) gid=1002(hwang) groups=44(video),1002(hwang)
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
# pwd
/home/hwang
$ su -
Password:
# id
uid=0(root) gid=0(root) groups=0(root)
# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
# pwd
/root
#

```

使用“-c”选项运行su命令时，普通用户能够以超级用户的身份及其访问权限执行特权命令，在命令运行结束后，立即返回原来的Shell工作环境。下面的例子表示普通用户无法使用kill命令终止一个进程，但当使用“su -c”命令并提供超级用户的密码之后，即可正常执行kill命令（注意，“-c”选项后面的命令、选项及其参数前后需加双引号）：

```

$ kill -15 5575
-bash: kill: (5575) - Operation not permitted
$ su -c "kill -15 5575"
Password:
$

```


第10章 进 程 管 理

进程是Linux系统中的重要概念。本章主要介绍怎样查询进程的状态信息，监控进程及系统资源，讨论怎样利用进程管理工具，找出耗时的无效进程，中止进程的运行，调整进程的优先级，使关键业务处理能够以较高的优先级运行，普通的例行业务能够以较低的优先级运行，从而提高系统的整体性能。

所谓的进程是指处于运行状态的程序。严格地说，进程是程序从开始调度运行直至终止执行的整个生命周期的全过程。进程管理是Linux系统中的一个重要组成部分，负责管理和控制所有的动态过程和资源（文件系统负责管理所有的静态信息和资源）。

通常，Linux系统中的进程可分为两大类：系统进程和用户进程。系统进程主要负责Linux系统的生成、管理、维护和控制，其中包括init进程等。用户进程指的是由用户通过Shell命令行界面（或GUI桌面）提交系统运行的命令和应用程序等。除了少数进程外，系统中的所有进程几乎都是由init进程直接或间接启动的。也就是说，init进程几乎是所有进程的直接或间接父进程。

每个进程都有一个系统赋予的进程标识（进程ID或PID），并与启动进程的用户（用户ID）等相关联，所有的进程统由进程子系统负责管理。用户可以查询进程的状态信息，但只能控制自己的进程，例如向进程发送控制信号，重新启动和终止进程等。只有超级用户才有权力控制所有的进程。

Linux系统采用多用户、多任务的进程管理机制，保证所有处于竞争状态的进程都能够公平合理地分享系统资源，保证重要的进程能够优先分配到资源，普通用户和超级用户均可程度不同地、实时地调整活动进程的优先级。

10.1 ps命令概述

在Linux系统中，获取进程状态信息的常用工具是ps命令。ps命令可用于查询系统中活动进程的状态信息，如进程的起始运行时间和资源占用情况等，这些信息对进程和系统性能的管理都是非常有用的。ps命令的语法格式简写如下：

```
ps [-aAcefHlW] [-g grouplist] [-p pidlist] [-t termlist] [-u userlist]
```

表10-1 给出了ps命令的部分常用选项。

表10-1 ps命令的常用选项

选项	GNU选项	描述
-a		显示系统中所有活动进程的当前状态信息（与终端无关联的进程除外）
-A		显示系统中当前所有进程的状态信息。其作用等同于“-e”选项
-c		与“-l”选项一起使用时能够显示进程的调度信息，包括进程的调度类别与优先级等

(续表)

选项	GNU选项	描述
-e		显示系统中当前所有进程的状态信息
-f		显示进程的重要状态信息，尤其是进程的起始运行时间和进程占用的CPU时间等
-F		与“-f”选项相比，能够显示更多重要的进程状态信息
-H		以表示进程调用层次关系的缩进形式显示所有进程的状态信息
	-forest	以表示进程调用层次关系的树形结构显示所有进程的状态信息
-l		显示进程的详细状态信息（进程的起始运行时间除外）
-w		显示完整的进程信息。在显示进程信息时，每个进程通常仅占一行，如果进程的信息较长，命令字段的超常部分将会被截断。使用“-w”选项时，超长部分将会延续到下一行输出
-g grouplist		显示与指定的有效用户组ID或用户组名有关的进程状态信息
-p pidlist		显示指定进程ID的进程状态信息
-t termlist		显示与指定的终端设备有关的进程状态信息
-u userlist		显示与指定的有效用户ID或用户名有关的进程状态信息

最简单的ps命令通常只能显示当前用户自己的进程状态信息。例如：

```
$ ps
  PID TTY          TIME CMD
 6126 pts/0        00:00:00 bash
 6410 pts/0        00:00:00 ps
$
```

如果想要了解进程的更多信息，可在ps命令中增加“-e”或“-A”选项，这意味着显示所有进程的状态信息，增加“-f”、“-l”或“-F”等选项，还会输出更多的、重要的状态信息。而且，也可以组合使用ps命令的各种选项，例如：

```
$ ps -ef
  UID          PID    PPID  C STIME TTY          TIME CMD
  root           1         0  0  10:22 ?           00:00:01 /sbin/init
  root           2         0  0  10:22 ?           00:00:00 [kthreadd]
  root           3         2  0  10:22 ?           00:00:00 [migration/0]
  root           4         2  0  10:22 ?           00:00:00 [ksoftirqd/0]
  root           5         2  0  10:22 ?           00:00:00 [watchdog/0]
  root           6         2  0  10:22 ?           00:00:00 [events/0]
  .....
  gqxing       6126       6120  0  10:54 pts/0        00:00:00 bash
  gqxing       6416       6126  0  11:18 pts/0        00:00:00 ps -ef
$
```

根据提供的命令选项，ps命令通常能够给出下列信息：

- 进程当前的工作状态（S）；
- 进程标识（PID）；
- 父进程标识（PPID）；

- 用户标识（UID）；
- 进程所处的调度级别（CLS）；
- 进程的优先权（PRI）；
- 进程的地址空间（ADDR）；
- 进程占用的内存（RSS）；
- 进程占用的CPU时间（TIME）。

表10-2描述了ps命令输出的若干字段，这些字段是否能够全部出现以及何时出现，取决于提供的命令选项。

表10-2 ps命令输出的字段信息一览

字段	简单说明
F	进程标志字段。下面是ps命令参考手册中提到的两个标志： 1：已经创建（fork），但尚未单独运行的（exec）的进程 4：用到超级用户特权的进程
S	采用下列字符表示进程的当前状态： S：因等待某一事件的完成而处于休眠状态的进程（进程可以中断） D：处于休眠状态，但不能中断的进程（通常为I/O进程） R：表示正在运行的进程，或处于运行队列，正在等待调度运行的进程。任何时刻，每个CPU只能有一个进程处于运行状态 X：进程已终止（通常见不到此类进程） Z：僵尸进程（处于彻底终止运行之前中间阶段的进程） W：已经完全导入磁盘交换区，没有任何页面仍位于内存中的进程（2.6.xx内核开始取消） T：由于跟踪或因作业控制信号导致进程中断执行而处于暂停运行状态的进程
UID	进程属主的有效用户ID
PID	进程的进程ID
PPID	进程的父进程ID
C	进程生命周期的CPU利用率（百分比），也即进程实际使用的CPU时间除以进程整个生命周期耗费的时间总量（包括等待时间）
CLS	进程的调度类别，如标准的分时进程调度类别TS等
PRI	进程的优先级。优先级的数字越大表示进程的优先级越高
NI	进程优先级的nice调整值（其范围为-20~19），用于调整进程的优先级
ADDR	proc进程结构的地址空间
SZ	进程核心映像（包括代码段、数据段以及栈空间）占用的物理内存页面的大小
WCHAN	因等待某一资源或事件的发生而使进程处于等待状态的内核函数的名字（“-”表示进程正在运行）
STIME	进程的起始运行时间。如果起始时间位于24小时之内，以“HH:MM”形式表示，如果超过24小时，则以“mmm dd”形式表示起始运行时间，其中mmm表示月（如英文月名所写的前三个字母），dd表示日
TTY	控制终端。表示进程（或其父进程）是从哪一个终端上启动运行的。如果这个字段是问号“？”，则表示进程与任何控制终端无关
TIME	从调度运行开始，进程迄今累计占用的CPU时间总和。以“[dd-]hh:mm:ss”形式表示
CMD	进程对应的命令或程序的名字。如果命令行过长，超出部分（包括选项或参数）将被截断，使得每个进程仅占用一行显示位置（除非使用“-w”选项）

10.2 查询进程及其状态信息

10.2.1 查询当前活动的进程

利用“ps -a”命令，可以显示当前系统中从终端运行或调用的所有活动进程及其状态信息。例如：

```
$ ps -a
  PID TTY          TIME CMD
 6126 pts/0        00:00:00 bash
 6256 pts/0        00:00:00 ps
$
```

从命令的输出结果可以看出，当前系统中直接从终端运行或调用的活动进程只有bash和ps，其进程ID分别为6126和6256，从终端/dev/pts/0上启动运行的。

10.2.2 查询系统中的所有进程

ps命令经常用于检查期望运行的进程是否已经启动，或准备终止的进程是否已经停止运行。为了获取系统中当前调度运行的所有进程及其状态信息，可以使用“ps -e”或“ps -A”命令。例如：

```
$ ps -e
  PID TTY          TIME CMD
    1 ?            00:00:02 init
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 migration/0
    4 ?            00:00:00 ksoftirqd/0
    5 ?            00:00:00 watchdog/0
    6 ?            00:00:00 events/0
.....
 6126 pts/0        00:00:00 bash
 6528 pts/0        00:00:00 ps
$
```

在上述输出信息中，TTY字段的问号“?”表示系统中的许多进程都是由系统直接调度运行的，并非源于某个特定的终端。

利用“ps -e”命令能够给出当前系统中所有进程信息的特点，可以查询某个特定进程是否已经启动，这是ps命令的一个重要的应用。例如，为了查询Apache服务器的守护进程apache2当前是否正在运行，可以使用下列组合命令：

```
$ ps -e | grep apache2
5580 ?            00:00:00 apache2
5581 ?            00:00:00 apache2
5584 ?            00:00:00 apache2
5588 ?            00:00:00 apache2
$
```

也可以使用改进的`pgrep`或`pidof`命令，查询指定进程的运行状态。如果指定的进程已经启动且正在运行，下列命令将会给出活动进程的PID：

```
$ pgrep vsftpd
5480
5486
5488
$ pidof mysqld
4765
$
```

10.2.3 显示进程的重要状态信息

系统管理员经常需要考察系统中的进程，监控并找出影响系统性能、危及系统安全的进程。使用`ps`命令的“-f”选项，可以获取进程的重要状态信息。例如：

```
$ ps -f
UID          PID  PPID  C  STIME TTY          TIME CMD
gqxing      6126   6120  0  10:54 pts/0        00:00:00 -bash
gqxing      6340   6126  0  10:59 pts/0        00:00:00 ps -f
$
```

从上面的输出结果可以看出，其中给出了很多重要的进程信息，包括用户标识、进程标识、父进程标识、进程起始运行时间和累计运行时间等。“-f”选项经常与“-e”选项一起使用，用于显示系统中所有进程的重要状态信息。这将是一个长长的进程列表，下面只是截取了其中的一部分：

```
$ ps -ef
UID          PID  PPID  C  STIME TTY          TIME CMD
root           1     0  0  10:22 ?           00:00:00 /sbin/init
root           2     0  0  10:22 ?           00:00:00 [kthreadd]
root           3     2  0  10:22 ?           00:00:00 [migration/0]
root           4     2  0  10:22 ?           00:00:00 [ksoftirqd/0]
root           5     2  0  10:22 ?           00:00:00 [watchdog/0]
root           6     2  0  10:22 ?           00:00:00 [events/0]
.....
gqxing      6120     1  0  10:54 ?           00:00:41 gnome-terminal
gqxing      6125   6120  0  10:54 ?           00:00:00 gnome-pty-helper
gqxing      6126   6120  0  10:54 pts/0        00:00:00 bash
gqxing      6255   6126  0  11:10 pts/0        00:00:00 ps -ef
$
```

利用`ps`命令可以监控系统中是否存在异常的进程。针对`ps`命令输出的`STIME`和`TIME`字段，可以重点检查进程的起始运行时间和迄今为止累计占用的CPU时间。除非系统进程，如果某个用户进程从启动至今无缘无故地耗费了大量的CPU时间，很有可能进程已处于无限循环状态。

10.2.4 显示进程的详细状态信息

利用`ps`命令的“-l”选项，可以列出每个活动进程的详细状态信息。例如：

```
$ ps -l
F S    UID      PID   PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S    1000    6126   6120  0  80   0  -  1609 wait   pts/0    00:00:00 bash
0 R    1000    6286   6126  0  80   0  -   536 -      pts/0    00:00:00 ps
$
```

利用ps命令的“-c”选项，可以列出每个活动进程所在的调度类别及其优先级：

```
$ ps -cl
F S    UID      PID   PPID  CLS PRI  ADDR SZ WCHAN  TTY          TIME CMD
4 S    1000    6126   6120 TS   19  -  1609 wait   pts/0    00:00:00 bash
0 R    1000    6296   6126 TS   19  -   536 -      pts/0    00:00:00 ps
$
```

如果再组合使用ps命令的“-e”选项，则可以列出所有进程所在的调度类别及其优先级等详细状态信息：

```
$ ps -ecl
F S    UID      PID   PPID  CLS PRI  ADDR SZ WCHAN  TTY          TIME CMD
4 S      0        1      0 TS   19  -   657 poll_s ?      00:00:03 init
1 S      0        2      0 TS   24  -     0 kthrea ?      00:00:00 kthreadd
1 S      0        3      2 FF  139  -     0 migrat ?      00:00:00 migration/0
1 S      0        4      2 TS   24  -     0 ksofti ?      00:00:00 ksoftirqd/0
5 S      0        5      2 FF  139  -     0 watchd ?      00:00:00 watchdog/0
1 S      0        6      2 TS   24  -     0 worker ?      00:00:00 events/0
.....
0 S    1000    6120      1 TS   19  - 13094 poll_s ?      00:00:02 gnome-terminal
0 S    1000    6125    6120 TS   19  -   474 unix_s ?      00:00:00 gnome-pty-helper
0 S    1000    6126    6120 TS   19  -  1587 wait   pts/0    00:00:00 bash
0 R    1000    6306    6126 TS   19  -   605 -      pts/0    00:00:00 ps
$
```

上述输出信息说明，正处于休眠状态的进程bash、其父进程是6120，且是在进入gnome-terminal终端窗口后启动的。进程占用了1478个内存页面，现正于wait函数处等待用户的输入。

10.2.5 显示进程间的调用关系

ps命令新增的“-forest”选项可以按照缩进形式，显示进程的调用层次关系，以及进程的状态信息：

```
$ ps -ef --forest
UID      PID   PPID  C STIME TTY          TIME CMD
.....
root     1113      1  0 14:35 ?          00:00:00 /usr/sbin/inetd
root     1624    1113  0 14:36 ?          00:00:00 \_ in.telnetd: bogon
root     1630    1624  0 14:36 pts/0    00:00:00 \_ login -h bogon -p
ggxing   1869    1630  0 14:37 pts/0    00:00:00 \_ -bash
ggxing   6405    1869  0 17:28 pts/0    00:00:00 \_ ps -ef --forest
root     1149      1  0 14:35 ?          00:00:00 /usr/sbin/vsftpd
nobody   1673    1149  0 14:37 ?          00:00:00 \_ /usr/sbin/vsftpd
ggxing   1704    1673  0 14:37 ?          00:00:00 \_ /usr/sbin/vsftpd
.....
ggxing   6377      1  0 17:24 ?          00:00:01 gnome-terminal
```

```
gqxing 6378 6377 0 17:24 ? 00:00:00 \_ gnome-pty-helper
gqxing 6379 6377 0 17:24 pts/1 00:00:00 \_ bash
$
```

10.2.6 pstree命令

Linux系统还提供一个pstree命令，可以采用树形缩进形式显示进程之间的调用关系。其语法格式简写如下：

```
pstree [-achlnpu] [-H pid] [pid | user]
```

表10-3 给出了pstree命令的部分常用选项及其简单说明。

表10-3 pstree命令的常用选项

选项	简单描述
-a	显示进程的命令行参数。如果进程已转储到交换区，将会在进程名的前后增加一对圆括号
-c	禁止压缩等痛的进程子树。通常，pstree命令将会尽可能地压缩进程子树的显示
-h	高亮度显示当前进程及其父进程
-H [pid]	高亮度显示指定的进程
-l	显示完整的进程信息。通常，超长部分将会被截断，以适应屏幕的显示宽度
-n	显示时按照进程ID，而不是进程的名字排序父进程
-p	在每一个进程名之后，以外加圆括号的形式输出进程的ID号。通常不输出进程的ID号
-u	如果进程的UID与其父进程的UID不同，则在进程名之后以增加圆括号的形式输出用户ID号或用户名

取决于是否指定了进程，进程树形结构的根或是指定的进程，或是init进程。运行pstree命令时，如果未指定任何进程，则从init进程开始，按照树形缩进形式显示所有的进程。如果指定了某个进程ID，则从指定的进程开始，按照树形缩进形式显示其后继的所有进程。此外，如果指定了用户名，则从某个进程开始显示指定用户拥有的所有进程的调用关系。示例如下：

```
$ pstree
init--+-NetworkManager
.....
      |-gnome-terminal--+-bash
      |                   |-gnome-pty-helpe
      |                   `-- {gnome-terminal}
.....
      |-hald---hald-runner--+-hald-addon-acpi
      |                   |-hald-addon-gene
      |                   |-hald-addon-inpu
      |                   |-hald-addon-leds
      |                   `--hald-addon-stor
.....
$
```

“gnome-terminal”等行表示用户运行了多个进程，其中包括gnome-terminal、bash和当前正在运行的pstree进程。

如果增加了“-p”选项，pstree命令还会给出每个进程的进程ID。例如：

```
$ pstree -p
init(1) +-NetworkManager(787)
.....
| -gnome-terminal(6377) +-bash(6379)
|                         | -gnome-pty-helpe(6378)
|                         `-- {gnome-terminal}(6380)
.....
| -hald(659) ---hald-runner(801) +-hald-addon-acpi(1004)
|                                 | -hald-addon-gene(946)
|                                 | -hald-addon-inpu(969)
|                                 | -hald-addon-leds(941)
|                                 `--hald-addon-stor(1002)
.....
$
```

如果指定了进程ID，则从指定的进程处开始，显示其所有后继进程之间的树形调用关系。例如：

```
$ pstree 6377
gnome-terminal +-bash
                | -gnome-pty-helpe
                `-- {gnome-terminal}

$ pstree -p 6377
gnome-terminal(6377) +-bash(6379)
                    | -gnome-pty-helpe(6378)
                    `-- {gnome-terminal}(6380)

$
```

10.3 监控进程及系统资源

前面介绍了怎样利用ps命令获取进程的各种状态信息。需要说明的是，ps命令的输出只是系统执行命令那一刻的快照信息。为了能够连续地观察进程的实时状态信息以及其他系统信息，可以使用更受欢迎的top命令，其语法格式简写如下：

```
top [-hv | -bcisS] [-d delay] [-n iterations] [-p pids]
```

其中，“-d delay”选项表示数据取样和屏幕刷新的时间间隔，默认的取样时间间隔为3秒。delay可以是一个整数，也可以是形如“ss.tt”的小数，时间单位为秒。

正如其名字所蕴含的那样，top命令仅仅列出系统中的前17个进程（取决于终端窗口的大小）及其他系统信息，并周期地进行更新。top通常会按照进程的CPU占用率，从高到低对每个进程进行排序。

通常，top每隔3秒更新一次数据，以反映系统的当前运行状态。top命令的典型输出形式如图10-1所示。

top命令输出的上半部分是系统运行状态的汇总信息，其中第一行包括系统的当前时间（11:56:21）、自启动以来系统的累计运行时间（1小时45分）、当前注册到系统中的用户数量（2个），以及系统的三个平均负载值（0.93、0.82和0.37）。第二行是有关进程的统计信息，

其中包括系统中现有进程的总数（174）、当前正在运行的进程数量（1）、处于休眠状态的进程数量（173）、暂停运行的进程数量（0），以及处于僵尸状态的进程数量（0）等。第三行是对CPU工作状态的分类统计，其中包括CPU处于用户模式（25.0）、系统模式（6.9）、空闲状态（60.5）以及等待I/O状态（7.6）等的百分比。第四行是内存使用情况的分类统计，其中包括系统配置的物理内存总量（379948KB）、已用内存的数量（364020KB）、空闲内存的数量（15928KB）和用做缓冲区的内存数量（21148KB）等。第五行是对交换区的分类统计，其中包括系统配置的交换区总量（843876KB）、已用交换区的数量（3384KB）、空闲交换区的数量（840492KB）和用做缓冲区的交换区数量（124976KB）等。表10-4给出了top命令上半部分的输出信息及简要说明。

```

top - 11:56:21 up 1:45, 2 users, load average: 0.93, 0.82, 0.37
Tasks: 174 total, 1 running, 173 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 6.9%sy, 0.0%ni, 60.5%id, 7.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 379948k total, 364020k used, 15928k free, 21148k buffers
Swap: 843876k total, 3384k used, 840492k free, 124976k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM   TIME+  COMMAND
 1245 root        20   0  202m  15m 7340  S   7.3   4.1   1:22.05 Xorg
 1961 gqxing      20   0  58628  22m 16m  S   5.6   6.1   0:11.63 gnome-panel
 1962 gqxing      20   0  99328  28m 18m  S   5.6   7.6   0:20.83 nautilus
 2009 gqxing      20   0  23268  3340 2732  S   2.6   0.9   0:01.03 gvfs-gdu-volume
 2625 root        20   0  2560  1016  620  S   2.3   0.3   0:00.07 mount.ntfs
 1989 root        20   0  5248  3020 2436  S   1.3   0.8   0:02.39 devkit-disks-da
 565  messageb   20   0  3168  1428  748  S   1.0   0.4   0:03.41 dbus-daemon
 2241 gqxing      20   0  2464  1196  884  R   1.0   0.3   0:37.48 top
 2014 gqxing      20   0  23548  9084 7468  S   0.7   2.4   0:20.89 geyes_applet2
 2220 gqxing      20   0  52404  14m 10m  S   0.7   3.9   0:10.13 gnome-terminal
 1085 root        20   0  8884  3392 2792  S   0.3   0.9   0:00.10 gdm-binary
 1241 mysql       20   0  141m  17m 5420  S   0.3   4.7   0:13.32 mysql
 1966 gqxing      20   0  51748  23m 10m  S   0.3   6.2   0:20.75 ubuntuone-clien
 1972 gqxing      20   0  41320  13m 9688  S   0.3   3.5   0:00.91 update-notifier
 1983 gqxing      20   0  17740  6724 5412  S   0.3   1.8   0:00.30 gdu-notificatio
 1999 root        20   0  7448  3716 2852  S   0.3   1.0   0:00.25 polkitd
    1 root        20   0  2632  1444 1120  S   0.0   0.4   0:01.93 init
  
```

图10-1 top命令主界面

表10-4 top命令上半部分的输出信息及简单说明

字段	简单说明
hh:mm:ss	第一行给出的是系统当前运行状态的汇总统计信息。其中的“hh:mm:ss”字段表示系统的当前时间，以小时、分和秒的形式，表示top命令给出的是此刻的系统状态信息
up	系统自启动以来迄今为止的运行时间
users	系统中现有的注册用户数量
load averages	其中给出了三个平均系统负载值。每个数值分别表示在最近1分钟、5分钟和15分钟内系统运行队列的平均长度
Tasks	<p>第二行是进程运行状态信息的汇总统计。其中第一个字段（total）表示系统当前共有多少个进程。其他字段分别表示处于下列每一种运行状态下的进程数量：</p> <ul style="list-style-type: none"> • running: 正在运行，或已处于运行队列，一旦调度即可运行的进程数量。注意，这个数字有可能会大于系统配置的CPU个数 • sleeping: 因等待外部事件完成而处于休眠状态的进程数量 • stopped: 因跟踪调试或暂时停止运行的进程数量 • zombie: 僵尸进程的数量
Cpu(s)	<p>第三行是CPU工作状态的分类统计信息，分述如下：</p> <ul style="list-style-type: none"> • %us: CPU处于用户模式的时间量所占的百分比 • %sy: CPU处于系统模式的时间量所占的百分比 • %ni: CPU处理其优先级经nice值调整过的用户进程的时间量所占的百分比 • %id: CPU处于空闲状态的时间量所占的百分比

(续表)

字段	简单说明
	<ul style="list-style-type: none"> • %wa: CPU等待I/O完成的时间量所占的百分比 • %hi: CPU处理硬件中断的时间量所占的百分比 • %si: CPU处理软件中断的时间量所占的百分比
Mem	<p>第四行是系统物理内存使用情况的汇总统计信息。其中每个字段的意义分述如下:</p> <ul style="list-style-type: none"> • total: 系统配置的可用物理内存总数 • used: 当前已经使用的物理内存计数 • free: 当前仍然空闲的物理内存计数 • buffers: 用做系统缓冲区的物理内存计数
Swap	<p>第五行是交换区使用情况的汇总统计信息。其中每个字段的意义分述如下:</p> <ul style="list-style-type: none"> • total: 系统配置的交换区总数 • used: 当前已经使用的交换区计数 • free: 当前仍然空闲的交换区计数 • cached: 用做系统缓存的交换区计数

从图10-1所示输出数据可以看出,截至11时56分21秒,在最近1、5和15分钟之内,系统的平均负载量分别是0.93、0.82和0.37。总共有174个进程,但除了top进程之外,其他大部分进程均处于休眠状态。CPU有60.5%的时间是空闲的,25.0%的时间直接服务于用户,6.9%的时间用于系统开销。通过这些数据,可以了解系统资源的使用情况。

top输出的第二部分根据每个进程的CPU占用量,按照从高到低的顺序,给出每个进程的状态信息。其中,PID是进程ID;USER是进程属主的用户名;PR是进程的优先级;NI是进程的nice调整值(范围从-20到20);VIRT是进程分配的整个虚拟内存空间(包括代码段和数据段等)的大小;RES是进程当前实际驻留的物理内存大小;SHR是进程占用的共享内存空间数量;S是进程的当前状态(D、R、S、T或Z);%CPU是进程占用CPU时间的百分比;%MEM是进程当前占用的物理内存的百分比;TIME(TIME+)是进程占用的系统和用户CPU时间的累计数量;COMMAND是当前进程对应的命令或程序(参见表10-5)。

表10-5 top提供的进程状态信息

字段	简单说明
PID	系统分配给每个进程的进程ID。进程ID通常是一个小于65536的正整数。一旦进程结束或被强行终止,进程ID还可以重用
USER	启动或拥有进程的用户名
PR	分配给每个进程的优先级,用于确定进程调度的优先顺序。Linux内核在计算和分配优先级时需要考虑许多因素,在进程的整个生命周期中,这个数值通常不会存在大的起伏变动
NI	进程优先级的nice调整值。nice调整值可以是负数,表示提高优先级。在Linux系统中,nice调整值的取值范围是-20至20。大多数用户进程采用0作为优先级的nice调整值,表示采用基本的优先级。但也可以选择使用一个大于零的nice调整值启动进程,使系统能够降低进程的优先级。对于一个需要长时间占用CPU(运行时间长,以CPU处理为主)的处理任务来讲,这是一种常规的做法,可以避免干扰交互式进程。但只有超级用户才能采用负数提高进程的优先级。Linux系统中的nice或renice命令可用于设置进程优先级的nice调整值

（续表）

字段	简单说明
VIRT	进程分配的虚拟内存映像空间数量（以KB为单位），是进程使用的虚拟内存总和，其中包括代码段、数据段和共享库占用的所有空间，以及转储到交换区的页面数量。进程的虚拟内存映像空间数量可用下式表示： $VIRT = SWAP + RES$
RES	进程占用的基本物理内存（即非交换内存）空间数量（以KB为单位）。一个进程可以申请分配较大的虚拟内存空间，但通常只能使用非常小的物理内存。进程的基本物理内存可用下式表示： $RES = CODE + DATA$
SHR	进程占用的共享内存空间数量（以KB为单位）。这是为进程分配的全部内存空间数量，也即整个虚拟内存空间的总量。这个数量等于进程的代码段、数据段或栈段等内存空间的总和
S	当前的进程状态信息。任何进程总是处于下列状态之一： <ul style="list-style-type: none">• D: 进程处于不可中断的休眠状态• R: 进程正在运行，或已处于运行队列，一旦调度即可运行• S: 进程因等待外部事件的完成而处于休眠状态• T: 进程因跟踪调试，或因收到某个信号（如Ctrl-Z）而暂时停止运行• Z: 进程已终止，但其父进程尚未完成善后处理
%CPU	进程占用CPU的百分比。top通常以此列数据基准，按照从大到小的顺序显示进程
%MEM	进程当前占用的物理内存的百分比
TIME/TIME+	进程累计占用的CPU时间（TIME以秒为单位，TIME+以1/100秒为单位）。表示自开始运行以来，迄今为止进程占用的CPU时间。这个时间量是运行相应进程时实际耗费的CPU时间
COMMAND	启动进程的命令行或程序名

实际上，top命令还可以给出更多的统计信息（只是受限于屏幕的显示宽度）。top采用一个由26个英文字母组成的字符串AbcdEfghIjKlMnOpQrsTuvWxyz，分别表示PID、PPID、RUSER、UID、USER、GROUP、TTY、PR、NI、P、%CPU、TIME、TIME+、%MEM、VIRT、SWAP、RES、CODE、DATA、SHR、nFLT、nDRT、S、COMMAND、WCHAN和Flags26个字段。在上述字符串中，大写字母表示输出相应的字段，且按字母的排列顺序输出相应的字段。为了显示其他字段信息，可以输入“f”字符命令，然后键入相应字段的小写字母，把原来的小写字母改为大写字母。为了改变输出字段的排列顺序，可以输入“o”字符命令，连续输入任何小写字母，可以把相应的输出字段依次后移一个位置；连续输入任何大写字母，可以把相应的输出字段依次前移一个位置。通过调整字母的排列位置，达到调整输出字段的先后顺序的目的，如图10-2所示。

表10-6 是对top命令提供的其他进程状态信息的简单说明。

表10-6 top提供的其他进程状态信息

字段	简单说明
SWAP	进程的虚拟内存映像转储到交换区部分的空间数量（以KB为单位）
CODE	可执行代码段占用的物理内存数量（以KB为单位）
DATA	除了可执行代码之外的数据段和栈占用的物理内存数量（以KB为单位）
nFLT	页面故障计数。当进程尝试读取或写入一个虚拟的内存页面，而其地址空间不存在相应的页面时，即会出现页面故障，导致系统访问磁盘

(续表)

字段	简单说明
n DRT	数据修改后尚未写入磁盘的内存页面计数。在能够把这些物理内存空间作为虚拟内存页面分配给其他进程之前, 必须把其中的数据写入磁盘
WCHAN	进程处于休眠状态所在的函数。取决于系统内核的编译与链接状况。这个字段通常会给出系统内核休眠时所处函数的名字或地址。对于正在运行的进程而言, 这个字段位置将会存在一个减号“-”标志
Flags	表示进程的当前调度标志。参见/usr/include/linux/sched.h文件和图10-2

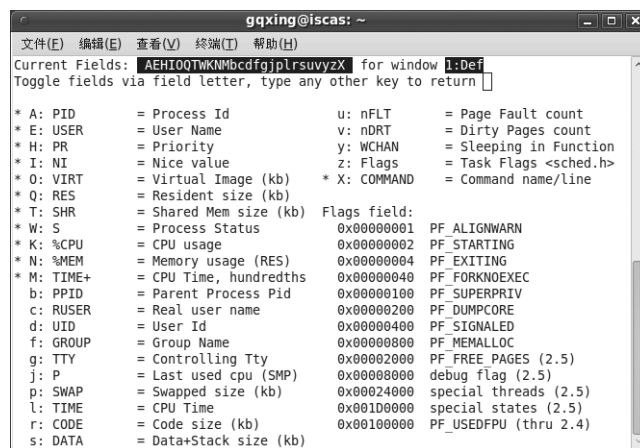


图10-2 top命令输出配置界面

top命令在输出各种统计信息, 不断刷新屏幕的同时, 也提供一个交互界面。利用top命令支持各种交互命令, 可以影响top命令的输出结果。例如, 输入“n”或“#”字符命令, 可以限定top能够显示的进程数量。输入“h”字符命令, 可以显示top命令交互操作的使用说明, 如图10-3所示。为了退出top命令, 可以直接输入“q”字符命令。

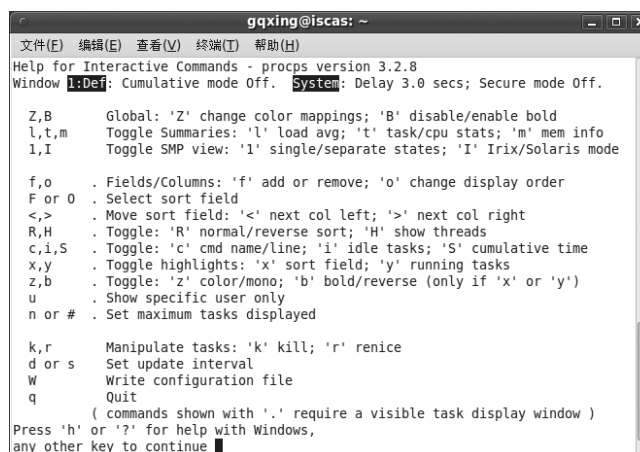


图10-3 top命令交互操作说明

10.4 终止进程的运行

有时，用户需要强行终止某个进程。例如，进程的运行时间过长，或进程因逻辑错误而陷于无限循环状态等。普通用户只能终止自己的进程，只有超级用户才能终止系统中的任何进程（进程标识号较小的内核进程除外，终止此类进程通常会引起系统瘫痪）。

Linux系统既支持POSIX标准信号，也支持POSIX实时信号。在收到信号之后，进程通常会采取下列默认的处理动作之一：

- 终止进程的运行；
- 忽略收到的信号，进程继续运行；
- 终止进程的运行，并把进程的内存映像转储到一个core文件；
- 暂时停止进程的执行。

为了强行终止一个进程，可以使用Shell的内部命令kill，或/bin/kill命令。kill命令的语法格式简写如下：

```
kill [-s sigspec | -n signal | -sigspec] pid ...  
kill -l [sigspec]
```

其中，sigspec既可以是一个规范的信号名（如SIGKILL），或缩写的信号名（如KILL），也可以是一个信号编码。signal是一个信号编码，表示kill命令发送给指定进程的信号（参见表10-7及/usr/include/bits/signal.h文件）。pid是进程ID。此外，利用“-l”选项还可以列出系统支持的所有信号（或指定信号）。

表10-7 Linux系统支持的部分信号

信号	信号名	默认的处理动作	简单说明
1	SIGHUP或HUP	终止进程	挂断。终端通信连接断开或控制进程终止时产生的信号
2	SIGINT或INT	终止进程	中断。按下中断键（也即Ctrl-C组合键）时产生的信号
3	SIGQUIT或QUIT	终止进程，生成内存映像文件（core）	退出。按下Quit键（也即Ctrl-\或Ctrl-Shift-组合键）时产生的信号
4	SIGILL或ILL	终止进程，生成内存映像文件（core）	非法指令。执行非法机器指令时产生的信号。其中包括由非法操作码、非法操作数、非法寻址模式、寄存器或内部栈错误等原因产生的信号
5	SIGTRAP或TRAP	终止进程，生成内存映像文件（core）	硬件故障或断点跟踪等情况产生的信号
6	SIGABRT或ABRT	终止进程，生成内存映像文件（core）	异常终止信号（由abort()函数产生的异常终止信号）
7	SIGBUS或BUS	终止进程，生成内存映像文件（core）	总线故障信号。包括由非法地址、不存在的地址或其他硬件错误等原因产生的信号
8	SIGFPE或FPE	终止进程，生成内存映像文件（core）	浮点算术运算异常（如除数为零或浮点运算溢出）时产生的信号
9	SIGKILL或KILL	终止进程	进程无法捕捉与封锁，也不能忽略的终止信号。可供系统管理员使用kill命令终止任何进程（部分核心进程除外）

(续表)

信号	信号名	默认的处理动作	简单说明
10	SIGUSR1或USR1	终止进程	用户定义的信号1, 供应用编程使用
11	SIGSEGV或SEGV	终止进程, 生成内存映像文件 (core)	内存地址越界或访问权限不足时产生的信号 (当代码地址超出进程的地址空间时产生的信号)
12	SIGUSR2或USR2	终止进程	用户定义的信号2, 供应用编程使用
13	SIGPIPE或PIPE	终止进程	管道断开信号 (当进程写入一个管道或套接字, 但读取管道或套接字的另一个进程已经终止时产生的信号)
14	SIGALRM或ALRM	终止进程	由alarm()系统调用产生的时钟超时信号
15	SIGTERM或TERM	终止进程	终止进程信号。这是kill命令产生的默认终止信号
16	SIGSTKFLT 或STKFLT	终止进程	栈故障信号
17	SIGCHLD或CHLD	忽略	子进程状态发生变动信号, 如子进程结束或停止运行时向父进程发送的信号
18	SIGCONT或CONT	继续 (或忽略)	令进程继续运行的信号, 主要用于作业控制。如果进程已暂停运行, 则默认的处理动作是继续运行。如果进程正在运行, 则默认的处理动作是忽略收到的信号
19	SIGSTOP或STOP	停止进程	停止进程运行信号。这是用于停止一个进程的作业控制信号, 这个信号主要用于作业控制, 既不能捕捉, 也不能忽略
20	SIGTSTP或TSTP	停止进程	键盘停止信号。在交互方式下, 通过按Ctrl-Z组合键停止当前进程时产生的信号。主要用于作业控制
21	SIGTTIN或TTIN	停止进程	后台进程试图从控制终端中读取数据时产生的信号。主要用于作业控制
22	SIGTTOU或TTOU	停止进程	后台进程试图向控制终端输出数据时产生的信号。主要用于作业控制
23	SIGURG或URG	忽略	当从网络连接 (套接字) 中接收到的数据发生错误, 通知进程出现紧急情况时发送的信号
24	SIGXCPU或XCPU	终止进程, 生成内存映像文件 (core)	当进程超过了其最大的软性CPU时间限制时产生的信号
25	SIGXFSZ或XFSZ	终止进程, 生成内存映像文件 (core)	当进程创建的文件超过了其能够创建的软性最大文件容量限制时产生的信号
26	SIGVTALRM 或VTALRM	终止进程	由setitimer()系统调用设置的虚拟间隔时钟超时信号
27	SIGPROF或PROF	终止进程	由setitimer()系统调用设置的内核抽样间隔时钟超时信号
28	SIGWINCH或 WINCH	忽略	窗口大小发生变动时产生的信号
29	SIGIO或IO	终止进程	异步I/O事件出现后产生的信号
30	SIGPWR或PWR	忽略	电源故障信号。当电源出现故障, 改由UPS提供系统电源供电时产生的信号。这个信号通常主要由操作系统处理, 应用程序不受影响, 除非UPS电源供电也不充足
31	SIGSYS或SYS	终止进程, 生成内存映像文件 (core)	系统调用有误。发现非法系统调用 (如参数有误) 时产生的信号

如果运行kill命令时未指定任何信号，则发送默认的信号15（SIGTERM）。在表10-5给出的信号中，信号9（SIGKILL）是不能捕捉的，因而可用于强行终止任何进程。也就是说，在kill命令中使用信号9，可以确保进程无条件终止。然而，信号9不能轻易使用。建议不要使用信号9终止诸如数据库服务器等进程，以免造成数据的丢失。

任何用户均可以终止自己的进程。如果想终止其他用户的进程，必须拥有超级用户的权限。为了终止进程的继续运行，通常可参照下列步骤：

首先，可以使用ps、pgrep或pidof命令获取进程的PID。例如，下列ps命令用于显示系统启动的所有telnetd守护进程：

```
$ ps -ef | grep telnetd | grep -v grep
root      6370   5072   0 14:38 ?        00:00:00 in.telnetd: winxp
root      6881   5072   0 18:14 ?        00:00:00 in.telnetd: sinosoft
$
```

此外还可以使用pgrep和pidof命令直接获取指定进程的PID。实际上，pgrep和pidof命令是对ps与grep组合命令的改进，可以直接给出指定进程的PID。例如，下列pidof的输出结果表示系统中当前共启动了4个apache2守护进程：

```
$ pidof apache2
6808 6807 6806 6803
$
```

下列pgrep命令的输出结果表示当前系统中共有3个注册Shell——bash在运行：

```
$ pgrep bash
1869
6379
6933
$
```

然后，可以使用kill（或pkill）命令终止进程的继续运行。例如，为了终止Telnet远程注册守护进程telnetd，禁止任何远程访问，可以使用下列kill命令：

```
$ sudo kill -9 6370 6881
$
```



注意 利用kill命令终止一个进程时，首先应尝试使用不带任何信号参数的kill命令，然后再观察进程是否已经终止。如果不奏效，再使用信号9。

最后，可以利用ps、pgrep或pidof命令验证进程是否已经终止。如果进程确已终止，ps、pgrep和pidof等命令不会输出任何结果。

10.5 调整分时进程的优先级

进程的优先级是由系统的进程调度程序根据调度策略分配的。用户可以利用nice或renice命令调整进程的优先级。

10.5.1 nice命令

nice命令是与UNIX系统兼容的进程优先级调整命令，renice命令是伯克利版renice命令的实现，具有更灵活的进程管理功能。一个进程的优先级是由进程所在调度类别的调度策略确定的，具体来讲是由进程的全局优先级与nice调整值共同确定的。实际上，确定进程的优先级涉及许多因素，其中包括进程调度类别，进程已经占用的CPU时间，以及在分时进程情况下的nice调整值等。

在开始运行之后，进程的优先级也可以通过nice或renice命令予以适当的调整。从下列ps命令输出的NI字段可以看到每个进程的优先级与nice调整值：

```
$ ps -el
 F S      UID      PID  PPID  C PRI  NI ADDR  SZ WCHAN  TTY          TIME CMD
 4 S        0        1      0  0  80   0 -    657 poll_s ?           00:00:03 init
 1 S        0        2      0  0  75  -5 -      0 kthrea ?           00:00:00 kthreadd
 1 S        0        3      2  0 -40   - -      0 migrat ?           00:00:00 migration/0
 1 S        0        4      2  0  75  -5 -      0 ksofti ?           00:00:00 ksoftirqd/0
 5 S        0        5      2  0 -40   - -      0 watchd ?           00:00:00 watchdog/0
 1 S        0        6      2  0  75  -5 -      0 worker ?           00:00:00 events/0
 .....
 0 S    1000    6120      1  0  80   0 -   13094 poll_s ?           00:00:17 gnome-terminal
 0 S    1000    6125    6120  0  80   0 -     474 unix_s ?           00:00:00 gnome-pty-help
 0 S    1000    6126    6120  0  80   0 -   1587 wait   pts/0       00:00:00 bash
 0 R    1000    7773    6126  0  80   0 -     605 -      pts/0       00:00:00 ps
$
```

通过增加nice调整值，普通用户可以降低一个进程的优先级。但只有超级用户才能通过减少nice调整值，达到增加进程优先级的目的。这个限制可防止用户增加自己的进程优先级，因而无止境地独占共享的CPU时间。对于超级用户而言，nice调整值的取值范围为-20~19，其中-20是可用的最大nice调整值。对于普通用户而言，nice调整值的取值范围为0~19，其中19是可用的最小nice调整值。

nice命令的语法格式简写如下：

```
nice [-n number] [command [arguments]]
```

其中，“-n”选项用于指定进程的nice调整值，如果未指定“-n”选项，默认的nice调整值是10。

按照上述说明，普通用户只能降低进程的优先级，而超级用户既可降低进程的优先级，又能提高进程的优先级。为了修改进程的优先级，针对普通用户与超级用户，分述如下。

作为一个普通用户，如果确实需要改变进程的优先级，使一个运行时间较长的普通应用程序能够以一个较低的优先级运行，以免影响其他业务的处理，可通过提高nice调整值，降低指定命令的优先级。

例如，下列nice命令通过增加5个nice调整值，以一个相对较低的优先级执行给定的命令（在未使用nice命令之前，find进程的优先级与nice调整值分别为80与0，使用nice命令之后，其优先级与nice调整值分别为85与5，find进程的优先级最终降低为85）：

```
$ find / -name core > /dev/null 2>&1 &
```



```
[1] 6445
$ ps -el | grep find
0 D 1000 6445 6377 12 80 0 - 595 sync_b pts/0 00:00:02 find
$ nice -n 5 find / -name core > /dev/null 2>&1 &
[2] 6448
$ ps -el | grep find
0 D 1000 6448 6377 8 85 5 - 549 sync_b pts/0 00:00:03 find
$
```

作为一个超级用户，如果确实需要改变进程的优先级，使一个关键业务处理能够以一个较高的优先级运行，而普通的应用能够以较低的优先级运行，可以通过降低或提高nice调整值，相应地提高或降低进程的优先级。

例如，下列nice命令通过降低10个nice调整值，相应地提高指定命令的优先级（在未使用nice命令之前，find进程的优先级与nice调整值分别为80与0，使用nice命令之后，其优先级与nice调整值分别为70与-10，find进程的优先级最终提高为70）：

```
$ sudo nice -n -10 find / -name core > /dev/null 2>&1 &
[1] 6478
$ ps -el | grep find
4 D 1000 6478 6377 8 70 -10 - 963 sync_b pts/0 00:00:01 find
$
```



如果赋予进程较高的nice调整值，有可能会过多地挤占其他进程的CPU时间，从而影响系统的整体性能。

10.5.2 renice命令

Linux系统提供的另外一个调整进程优先级的命令是renice，可用于调整正在运行的一个或多个进程的优先级。renice命令的语法格式简写如下：

```
renice priority [[-p] pids] [[-u] users]
```

其中，*priority*是一个整数数值，表示赋予进程的nice调整值，正数表示降低进程优先级的nice调整值，负数表示增加进程优先级的nice调整值，0表示保持进程的基本优先级不变。“-p”选项用于指定进程的进程ID。“-u”选项用于指定属于某个特定用户的所有进程。

同样，在调整进程的优先级时，普通用户只能利用renice命令增加nice调整值（其范围为0~20），从而降低进程的优先级。超级用户可以利用renice命令，调整任何进程的优先级，增加或减少nice调整值，其范围为-20~20。如果把nice调整值设定为20，则仅当系统中没有其他可以调度运行的进程时，才会调度执行renice命令指定的进程。采用负数增加进程的优先级时，进程能够得到更多的调度机会，从而运行得更快。nice调整值-20是超级用户可用的最大优先级。

例如，下列命令可以改变进程ID号为7040和7043，以及用户www-data拥有的所有进程的优先级：

```
$ sudo renice +1 -p 7040 7043 -u www-data
7040: old priority 0, new priority 1
7043: old priority 0, new priority 1
33: old priority 0, new priority 1
$
```


10.5.3 调整进程优先级的作用

下面，我们将利用time命令，分别采用提高或降低进程优先级nice调整值的方式运行同一命令，观察修改nice调整值前后的运行效果。

在此之前，首先简单介绍一下time命令。time命令可用于考察一个命令的执行效率。当利用time命令提交的程序运行结束之后，time命令将会从三个方面，给出在运行指定的程序时占用的时间：

- real: 表示运行给定程序时耗费的全部时间（以秒为单位）。这个时间包括从调用程序开始，直至程序执行结束期间的全部时间。
- user: 程序的实际运行时间（以秒为单位）。这个时间仅包含系统执行用户代码时占用的时间，而不包括操作系统执行系统调用等开销。
- sys: 表示在系统内核地址空间中执行代码期间占用的时间（以秒为单位）。这个时间包括内存分配、系统调用，以及数据I/O等开销。

现在让我们回过头来考察在提高或降低进程nice调整值前后，执行同一个命令时有何变化。下面的例子是在提高10个nice调整值时，运行dd命令的结果：

```
$ sudo time nice -n -10 dd if=/dev/sda8 of=/dev/null
23768577+0 records in
23768577+0 records out
12169511424 bytes (12 GB) copied, 1676.81 s, 7.3 MB/s

real    27m56.990s
user    0m40.333s
sys     3m34.508s
$
```

下面的例子是在降低10个nice调整值时，运行dd命令的结果：

```
$ sudo time nice -n 10 dd if=/dev/sda8 of=/dev/null
23768577+0 records in
23768577+0 records out
12169511424 bytes (12 GB) copied, 2124.78 s, 5.7 MB/s

real    35m25.154s
user    0m39.044s
sys     3m31.094s
$
```

从上面两个以不同进程优先级运行dd命令的输出结果可以看出两点：一是在程序的实际执行时间方面并无太大的差别。这是可以理解的，因为执行的是同一个程序，其代码和处理内容完全一样。二是两者在执行程序时耗费的全部时间方面差别却非常之大，这是因为具有不同优先级的进程在系统的进程调度方面得到的运行机会不同而造成的。进程的优先级越低，得到调度运行的机会就越少；反之，得到调度运行的机会就越多。

nice和renice命令的主要用途是降低一些运行时间较长，但又并非紧急事务的进程，使之能够以较低的优先级运行，提高系统对关键业务的响应速度，从而达到提高整体系统性能的目的。

第11章 proc文件系统

proc文件系统是一个虚拟文件系统，不占用任何磁盘空间。proc文件系统是系统运行状况的动态反映，可以用做系统内核数据结构的接口，以便用户获取进程状态以及可调参数等信息，而不必直接读取或解释/dev/kmem核心内存文件。proc文件系统通常安装在/proc目录下，其中的大多数文件只能读，部分与系统内核变量有关的文件允许修改，借以调整系统内核的参数。

proc文件系统目录或文件大体上可以分为三部分：一部分是系统当前所有进程的内存映像；第二部分是系统当前的配置与运行状态信息，第三部分是系统内核的可调参数。

11.1 进程内存映像文件

在/proc目录中，凡是以数字形式命名的目录均为进程的内存映像。系统中当前正在运行的每一个进程，均存在一个对应的目录，目录的名字以相应进程的进程ID命名。其中，目录1对应于著名的init进程。

在以进程ID命名的每个目录中，存在一系列虚拟文件或目录，如cmdline、cwd、environ、exe、fd、fdinfo、limits、maps、mem、root、smaps、stat、statm、status和task等文件，这些文件可用于考察进程的运行状态。下面以进程1，即init进程为例，简单介绍其中的部分重要文件。

cmdline文件包含进程的完整命令行信息（除非进程已经交换到swap区或处于zombie状态，cmdline文件为空），如命令的名字、选项和参数等，例如：

```
$ cat /proc/1/cmdline
/sbin/init$
```



在/proc目录下的文件中，一个文件可能包含多个环境变量（或命令行参数），每个变量只占一个字段，采用内部格式表示变量的字符串值，以字符“\0”作为字段分隔符或结尾。因此，为了使文件中的字符串数据可读性更强，可以使用“od -c”或“tr “\0” “\n””命令形式显示相应的文件。此外，也可使用“echo `cat <file>`”命令形式显示文件的内容。

cwd是一个符号链接文件，指向进程的当前工作目录。例如，为了找出进程1的工作目录，可以使用下列命令（注意，不能使用sudo直接运行cd命令，故在运行下列cd命令时可首先运行“sudo -i”命令，进入超级用户的Shell环境。此外，采用Shell内部命令pwd会产生错误的结果，故这里采用外部的pwd命令）：

```
$ sudo -i
[sudo] password for gqxing:
# cd /proc/1/cwd; /bin/pwd
/
#
```

environ文件包含进程运行环境的变量设置。由于其中的每一个字段中间均以“\0”字符作为分隔符，且最后仍以“\0”字符结束，因此为了查询进程1的运行环境，可以使用下列命令：

```
$ sudo cat /proc/1/environ | tr "\0" "\n"
ROOTFSTYPE=
HOME=/
DPKG_ARCH=i386
IPOPTS=
init=/sbin/init
.....
quiet=y
BOOT_IMAGE=/boot/vmlinuz-2.6.31-14-generic
PATH=/sbin:/usr/sbin:/bin:/usr/bin
.....
PWD=/
readonly=y
rootmnt=/root
ROOT=/dev/disk/by-uuid/472aa772-7de1-4b75-8c49-706973e9a213
BOOT=local
$
```

exe是一个符号链接文件，其中包含进程命令文件的实际路径名，引用这个文件相当于执行相应的命令。对于进程1而言，则相当于执行/sbin/init命令。

fd是一个子目录，其中包含进程打开的每一个文件，以文件描述符命名，其中0表示标准输入，1表示标准输出，2表示标准错误输出等。fdinfo是从Linux内核2.6.22版开始新增的一个目录，其中也包含进程打开的、以文件描述符命名的每一个文件，每个文件通常包含两个字段：pos字段是一个十进制的数字，表示文件读写的偏移值，flags字段是一个八进制的数字，表示文件的访问权限和文件状态（参见open(2)系统调用）。这个文件仅限于进程属主才能读取其中的内容。例如：

```
$ sudo ls -l /proc/1/fd
总计 0
lrwx----- 1 root root 64 2009-11-21 20:18 0 -> /dev/console
lrwx----- 1 root root 64 2009-11-21 20:18 1 -> /dev/console
lrwx----- 1 root root 64 2009-11-21 20:18 10 -> socket:[4128]
lrwx----- 1 root root 64 2009-11-21 20:18 2 -> /dev/console
lr-x----- 1 root root 64 2009-11-21 20:18 3 -> pipe:[2504]
l-wx----- 1 root root 64 2009-11-21 20:18 4 -> pipe:[2504]
lr-x----- 1 root root 64 2009-11-21 20:18 5 -> inotify
lr-x----- 1 root root 64 2009-11-21 20:18 6 -> inotify
lrwx----- 1 root root 64 2009-11-21 20:18 7 -> socket:[2505]
lrwx----- 1 root root 64 2009-11-21 20:18 8 -> socket:[3599]
lrwx----- 1 root root 64 2009-11-21 20:18 9 -> socket:[2560]
$ sudo cat /proc/1/fdinfo/4
pos:      0
flags:    04001
$
```

limits文件包含进程的各种资源的软性限制、硬性限制及其度量单位等，如CPU时间限制、创建文件的容量限制、创建进程的数量限制，以及打开文件的数量限制等。这个文件仅限于进程属主才能读取其中的内容。例如：

```
$ sudo cat /proc/1/limits
Limit                      Soft Limit                Hard Limit                Units
Max cpu time               unlimited                 unlimited                 ms
Max file size              unlimited                 unlimited                 bytes
Max data size              unlimited                 unlimited                 bytes
Max stack size             10485760                 unlimited                 bytes
Max core file size         0                        unlimited                 bytes
Max resident set           unlimited                 unlimited                 bytes
Max processes              3072                     3072                     processes
Max open files             1024                     1024                     files
.....
$
```

maps文件包含当前映射的内存区及其访问权限，例如：

```
$ sudo cat /proc/1/maps
00110000-0024e000    r-xp  00000000  08:02  5748    /lib/tls/i686/cmov/libc-2.10.1.so
0024e000-00250000    r--p  0013e000  08:02  5748    /lib/tls/i686/cmov/libc-2.10.1.so
00250000-00251000    rw-p  00140000  08:02  5748    /lib/tls/i686/cmov/libc-2.10.1.so
00251000-00254000    rw-p  00000000  00:00  0
003dc000-003dd000    r-xp  00000000  00:00  0        [vdso]
.....
00c30000-00c58000    r-xp  00000000  08:02  6074    /sbin/init
00c58000-00c59000    r--p  00028000  08:02  6074    /sbin/init
00c59000-00c5a000    rw-p  00029000  08:02  6074    /sbin/init
.....
01608000-0164a000    rw-p  00000000  00:00  0        [heap]
b7832000-b7835000    rw-p  00000000  00:00  0
b7843000-b7844000    rw-p  00000000  00:00  0
bfa52000-bfa67000    rw-p  00000000  00:00  0        [stack]
$
```

其中，第1个字段是进程占用的地址空间。第2个字段是由r（读）、w（写）、x（执行）、s（共享）和p（专用）等字符表示的访问权限。第3个字段是相对于文件起始位置的读写偏移值。第4个字段是以主次设备号“major:minor”形式表示的设备。第5个字段是文件所在文件系统中的信息节点号（0表示不存在与内存区相关联的信息节点，如bss）。第6个字段是进程的路径文件名。

mem文件反映的是进程的内存页面，可以利用open()、read()和fseek()等系统调用访问mem文件，进而访问进程的内存页面。

root是一个符号链接文件，指向进程的虚拟根目录。虚拟根目录是Linux系统的重要功能特性，利用chroot命令或chroot(2)系统调用，可以针对每个进程设置文件系统的虚拟根目录。在下面的例子中，进程1（也即init进程）的虚拟根目录仍为系统的实际根目录：

```
$ sudo ls -l /proc/1/root
lrwxrwxrwx 1 root root 0 2009-11-21 20:03 /proc/1/root -> /
$
```

smaps文件反映了每个进程的内存映射的使用情况。在每个进程的内存映射中，第一行输出信息与maps文件的内容完全相同。其他行分别给出了进程内存映射的大小、当前驻留内存部

分的大小、共享页面中已处理和尚未写回磁盘的内存大小，以及专用页面中已处理和尚未写回磁盘的内存大小等。例如：

```
$ sudo cat /proc/1/smmaps
.....
00c30000-00c58000 r-xp 00000000 08:02 6074      /sbin/init
Size:                160 kB
Rss:                 124 kB
Pss:                 124 kB
Shared_Clean:         0 kB
Shared_Dirty:         0 kB
Private_Clean:       124 kB
Private_Dirty:        0 kB
Referenced:           120 kB
Swap:                 0 kB
KernelPageSize:       4 kB
MMUPageSize:          4 kB
.....
$
```

stat文件包含进程的运行状态信息。实际上，ps命令也是利用这个文件输出进程状态信息的。文件内容的第1个字段是进程ID，位于圆括号中的第2个字段是进程的文件名，第3个字段采用一个字符表示进程的当前状态，其中包括R（运行）、S（休眠）、D（等待I/O）、Z（僵尸）、T（跟踪或停止）以及W（页面调度）等，第4个字段是父进程的进程ID，等等（参见ps命令的输出结果）。例如：

```
$ cat /proc/1/stat
1 (init) S 0 1 1 0 -1 4194560 4168 386135 20 1125 42 148 20227 1572 20 0 1 0 5 2588672 340
4294967295 12779520 12942900 0 0 0 0 0 4096 671163939 4294967295 0 0 0 0 0 0 0 0
$
```

statm文件以页面为单位，提供内存的状态信息——从左边第1个字段开始，其输出信息分别为：整个程序的大小、程序内存驻留部分的大小、共享页面数量、代码部分的大小、库函数部分的大小、数据或栈段部分的大小，以及尚未写入磁盘的数据页面数量。例如：

```
$ cat /proc/1/statm
632 340 270 40 0 99 0
$
```

status文件提供的信息主要来自/proc/pid/stat和/proc/pid/statm文件，只是以更容易阅读和理解的形式给出，其中包括进程的名字、状态（如R表示运行，S表示休眠等）、PID、有效用户ID、用户组ID、当前分配的文件描述符数量（FDSize），以及各种内存使用情况的详细数据，如虚拟内存大小（VmSize）、内存驻留部分的大小（VmRSS）、共享库代码部分的大小（VmLib）、代码段（VmExe）、数据段（VmData）及栈段（VmStk）的大小等。例如：

```
$ cat /proc/1/status
Name:   init
State:  S (sleeping)
Tgid:   1
Pid:    1
```

```

PPid:      0
TracerPid: 0
Uid:       0      0      0      0
Gid:       0      0      0      0
FDSize:    32
Groups:
VmPeak:    2532 kB
VmSize:    2528 kB
VmLck:     0 kB
VmHWM:     1492 kB
VmRSS:     1360 kB
VmData:    312 kB
VmStk:     84 kB
VmExe:     160 kB
VmLib:     1888 kB
VmPTE:     28 kB
Threads:   1
.....
$

```

task是一个目录。对于一个多线程的进程，其中可能包含若干子目录，每个子目录对应进程的一个线程。在每一个子目录中，也存在一组与进程目录中同名的文件，且其意义完全相同。

11.2 系统配置信息

通过查询proc文件系统，可以了解当前系统硬件设备的配置信息，如CPU、内存、系统总线、I/O设备、IDE和SCSI磁盘设备、磁盘分区、硬件中断、I/O端口的内存映射以及硬件设备的主次设备号等。

1. /proc/cpuinfo文件

/proc/cpuinfo文件含有CPU以及其他系统结构信息，包括CPU的数量、型号、缓存以及时钟频率等。例如：

```

$ cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 8
model name    : Pentium III (Coppermine)
stepping      : 10
cpu MHz       : 696.978
cache size    : 256 KB
.....
fpu           : yes
.....
$

```

其中的部分内容说明如下：

- processor——提供系统配备的每个CPU的标识号。在只有一个CPU的系统中，其标识号为0。

- `cpu family`——CPU系列的标识号。对于Intel x86 系列的计算机系统而言, 把这个数字加到“86”前面, 即可确定CPU的系列类型。在老式的586、486甚至386系统中, 这个数值是特别有用的。有些软件包是针对特定的CPU结构编译的, 因此, 这个数值有助于用户正确选择安装的软件包。
- `modelname`——CPU的通用名字。
- `cpu MHz`——CPU的时钟频率 (MHz), 精确到小数点后三位。
- `cache size`——CPU二级缓存的大小。
- `fpu`——浮点运算单元。
- `flags`——说明CPU的品质, 如是否包括浮点运算单元 (FPU), 以及是否具有处理MMX指令的能力等。

2. `/proc/filesystems` 文件

`/proc/filesystems` 文件包含系统内核当前支持的文件系统列表, 表示相应的文件系统支持模块已编译进系统内核。在安装文件系统时, 如果未指定文件系统的类型, `mount` 命令将会依次使用这个文件中的信息确定文件系统的类型, 尝试安装文件系统。例如:

```
$ cat /proc/filesystems
nodev    sysfs
nodev    rootfs
nodev    bdev
nodev    proc
.....
nodev    sockfs
nodev    usbfs
nodev    pipefs
nodev    anon_inodefs
nodev    tmpfs
.....
        ext3
        ext2
        ext4
nodev    ramfs
.....
$
```

在上述输出信息中, 第一列表示文件系统是否位于块设备中, 如果第一列为`nodev`, 意味着相应的文件系统并非基于磁盘设备的文件系统; 第二列给出的是Linux系统支持的文件系统的名字。

`/proc/iomem` 文件包含I/O内存映射, 例如

```
$ cat /proc/iomem
00000000-00001fff : System RAM
00002000-00005fff : reserved
00006000-00009fff : System RAM
00009f80-0000ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000cbfff : Video ROM
000dc000-000fffff : reserved
```

```

000e0000-000effff : Extension ROM
000f0000-000fffff : System ROM
00100000-17fdffff : System RAM
00100000-00575553 : Kernel code
00575554-0078d307 : Kernel data
0081a000-008a809f : Kernel bss
.....
$

```

3./proc/partitions文件

/proc/partitions文件包含磁盘与磁盘分区的主次设备号、存储容量（数据块数量）以及设备文件名。其内容类似于“fdisk -l”命令的输出结果。

```

$ cat /proc/partitions
major    minor    #blocks name
      8         0   29302560 sda
      8         1   13285723 sda1
      8         2   14353605 sda2
      8         3    843885 sda3
      8         4    819315 sda4
$ sudo fdisk -l /dev/sda

Disk /dev/sda: 30.0 GB, 30005821440 bytes
255 heads, 63 sectors/track, 3648 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0xc4cdcccd

   Device Boot      Start   End  Blocks  Id System
/dev/sda1  *           1   1654   13285723+  7 HPFS/NTFS
/dev/sda2             1655   3441   14353605  83  Linux
/dev/sda3             3441   3546    843885  82  Linux swap / Solaris
/dev/sda4             3547   3648    819315  1c  Hidden W95 FAT32 (LBA)
$

```

在上述partitions文件的输出信息中，每个字段的意义简述如下：

- major——磁盘以及磁盘分区的主设备号。
- minor——磁盘以及磁盘分区的次设备号。
- #blocks——磁盘以及磁盘分区中包含的物理磁盘数据块数量。
- name——磁盘以及磁盘分区的设备名。

4./proc/modules文件

/proc/modules文件包含已经加载到系统内核的所有模块列表，其内容等同于lsmod命令的输出结果。下面是modules文件中的部分内容，以及lsmod命令的部分输出结果：

```

$ cat /proc/modules
savage          30620      2  -      Live      0xd9129000
drm              159584     3  savage, Live      0xd90f4000
binfmt_misc      8356       1  -      Live      0xd90bb000
thinkpad_acpi    67108      0  -      Live      0xd909d000
snd_cs46xx       79488      2  -      Live      0xd9027000
.....

```



```
$ lsmod
Module          Size      Used by
savage          30620      2
drm             159584     3 savage
binfmt_misc     8356       1
thinkpad_acpi  67108      0
snd_cs46xx      79488      2
.....
$
```

在上述输出信息中，第一列是模块的名字。第二列是模块占用的内存字节数。第三列表示当前已经加载了多少个模块的实例（0表示模块尚未加载）。第四列表明相应模块是否依赖于其他模块的存在才能正常运行。如是，则列出依赖的模块。第五列是模块的加载状态，如Live、Loading或Unloading。第六列是加载的模块相对于当前系统内核起始内存地址的偏移值（这个信息可用于调试模块）。

5./proc/mounts文件

/proc/mounts文件能够提供系统当前安装的所有文件系统信息，其输出数据类似于/etc/mtab文件的内容，只是其数据总是保持最新状态（实际上，/proc目录中的所有文件总是保持最新状态）。这个文件的数据格式完全遵循fstab(5)文件的格式规定，示例如下：

```
$ cat /proc/mounts
rootfs / rootfs rw 0 0
none /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
none /proc proc rw,nosuid,nodev,noexec,relatime 0 0
udev /dev tmpfs rw,relatime,mode=755 0 0
/dev/disk/by-uuid/472a..... / ext4 rw,relatime,errors=remount-ro,barrier=1,
data=ordered 0 0
.....
$
```

在上述输出信息中，第一列是安装的设备名，第二列是安装点，第三列是文件系统的类型，第四列表示安装的方式，如只读（ro）或读写（rw）等。第五列和第六列并没有实际的意义，其目的只是为了保持与/etc/mtab文件格式一致。

6./proc/version文件

/proc/version文件含有当前运行的Linux系统内核（包括gcc）的版本信息。其中也提供了操作系统类型（参见/proc/sys/kernel/ostype文件）及版本（参见/proc/sys/kernel/osrelease与/proc/sys/kernel/version文件）等信息。例如

```
$ cat /proc/version
Linux version 2.6.31-14-generic (buildd@rothera) (gcc version 4.4.1 (Ubuntu
4.4.1-4ubuntu8) ) #48-Ubuntu SMP Fri Oct 16 14:04:26 UTC 2009
$ cat /proc/sys/kernel/ostype
Linux
$ cat /proc/sys/kernel/osrelease
2.6.31-14-generic
$ cat /proc/sys/kernel/version
#48-Ubuntu SMP Fri Oct 16 14:04:26 UTC 2009
$
```

11.3 系统运行状态信息

1./proc/cmdline文件

/proc/cmdline 文件含有启动系统时传递给Linux内核的引导参数，例如：

```
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-2.6.31-14-generic root=UUID=472a..... ro quiet splash
$
```

上述输出信息表示系统内核位于UUID表示的磁盘分区（即/dev/sda2，参见/etcfstab文件），详见第15章“系统启动与关机”。

2./proc/fs/nfsd/exports文件

/proc/fs 目录包含对外公布的共享文件系统，如果系统正在运行NFS服务器软件，使用cat命令显示/proc/fs/nfsd/exports文件，可以看到当前系统提供的共享文件系统，以及赋予文件系统的访问权限。例如：

```
$ cat /proc/fs/nfsd/exports
# Version 1.1
# Path Client (Flags) # IPs
/home/gqxing/docs iscas(rw,root_squash,sync,wdelay,no_subtree_check,uuid=7
3e5dfeb:ae7a4aae:9cfd6525:cca11ba2)
$
```

3./proc/kallsyms文件

/proc/kcore文件含有系统内核声明的各种模块定义，以便模块处理工具能够动态地连接与加载。

4./proc/kcore文件

/proc/kcore文件是系统物理内存的映像，采用ELF文件格式存储。利用这个虚拟文件与完整的二进制内核文件（/usr/src/linux/vmlinux），可以使用gdb等工具考察任何内核数据结构的当前状态。文件的大小等于物理内存的大小再加上4KB，以字节为单位。

5./proc/kmsg文件

/proc/kmsg文件用于存储系统内核生成的各种信息。注意，只有超级用户才能读取这个文件，且同时只能有一个进程读取这个文件。因此，如果采用syslog(2)系统调用读取并记录系统内核信息的syslog进程正在运行，则不应当读取这个文件，否则将处于等待状态，因而读不到任何数据。通常应使用dmesg命令读取这个文件中的系统信息。

6./proc/loadavg文件

/proc/loadavg文件的前3个字段是最近1分钟、5分钟以及15分钟之内系统负载的平均值。所谓的系统负载指的是位于运行队列（状态为R）或等待I/O完成（状态为D）的进程数量。这3个字段值与uptime命令输出结果中的最后3个系统负载平均值是相同的。第4个字段包括一对数值，中间增加一个斜杠“/”分隔字符，其中的第一个数值是系统中当前调度执行的进程或线程的数量，这个数值应小于或等于系统配置的CPU数量，第二个数值是系统当前调度运行的进程或线程的总数。第5个字段是系统中最近一次创建的进程ID。例如：

```
$ cat /proc/loadavg
0.33 0.26 0.10 2/292 2772
$ uptime
15:08:54 up 4:27, 3 users, load average: 0.33, 0.26, 0.10
$
```

7./proc/meminfo文件

/proc/meminfo文件提供了系统中整个内存、空闲内存和已用内存（包括实际内存和虚拟内存）的数量，以及系统内核使用的共享内存和缓冲区的数量。事实上，这个文件的内容与free命令的输出结果完全一致，不过free命令给出的只是其中的一部分。示例如下：

```
$ cat /proc/meminfo
MemTotal:      379948 kB
MemFree:       7156 kB
Buffers:       49836 kB
Cached:        119512 kB
SwapCached:    4872 kB
Active:        132504 kB
Inactive:      199432 kB
Active(anon):  61652 kB
Inactive(anon): 104764 kB
Active(file):  70852 kB
Inactive(file): 94668 kB
.....
SwapTotal:     843876 kB
SwapFree:      827836 kB
Dirty:         4 kB
Writeback:     0 kB
.....
VmallocTotal:  638968 kB
VmallocUsed:   3104 kB
VmallocChunk:  628724 kB
.....
$ free
```

	total	used	free	shared	buffers	cached
Mem:	379948	372792	7156	0	49852	119512
-/+ buffers/cache:		203428	176520			
Swap:	843876	16040	827836			

```
$
```

由于其中提供了大量有价值的系统内存及其使用情况，这个文件是/proc目录最常用的文件之一，free、top和ps等命令的许多输出信息大都取自这个文件。

- Mem Total——系统配备的物理内存的总和（以KB为单位）。
- Mem Free——系统中空闲物理内存的数量（以KB为单位）。
- Buffers——系统中用做文件缓冲区的物理内存数量（以KB为单位）。
- Cached——系统中用做缓冲区内存的物理内存数量（以KB为单位）。
- Swap Cached——用做缓冲区内存的交换区的数量（以KB为单位）。
- Active——当前正在使用的缓冲区或页面内存（即最近一直在用，通常无法回收利用的内存）的总和（以KB为单位）。

- Inactive——空闲可用的缓冲区或页面内存（即最近一直没有使用，因而可以回收利用的内存）的总和（以KB为单位）。
- HighTotal——没有直接映射为系统内核空间的内存总和（以KB为单位）。
- HighFree——非系统内核映射空间中的空闲内存数量（以KB为单位）。
- LowTotal——直接映射为系统内核空间的内存总和（以KB为单位）。
- LowFree——系统内核映射空间中的空闲内存数量（以KB为单位）。
- SwapTotal——可用交换区的容量总和（以KB为单位）。
- SwapFree——空闲交换区的容量总和（以KB为单位）。
- Dirty——其中的数据正等待写入磁盘的内存总和（以KB为单位）。
- Writeback——其中的数据正在写入磁盘的内存总和（以KB为单位）。
- Mapped——使用mmap命令映射设备、文件或库函数后占用的内存总和（以KB为单位）。
- Slab——系统内核使用的数据结构缓冲区占用的内存总和（以KB为单位）。
- VMallocTotal——分配用做虚拟地址空间的内存总和（以KB为单位）。
- VMallocUsed——已用于虚拟地址空间的内存总和（以KB为单位）。
- VMallocChunk——虚拟地址空间可用的最大连续内存块（以KB为单位）。

8./proc/net子目录

/proc/net子目录存在各种网络文件，这些文件给出了每个网络层的状态信息。文件的内容均为普通的ASCII数据，因而可以利用cat等命令进行查询。实际上，netstat和ifconfig等命令就是利用这些文件作为其信息源，提供各种统计数据的。而且，netstat等命令提供的数据可读性更强，显示格式更规范，故这里仅以两个文件为例予以说明。

(1) /proc/net/arp文件

/proc/net/arp文件包含系统内核实现地址解析的ARP表。其中的ARP解析项或者是预定义的，或者是动态获取的。在下述输出信息中，有关“HW type”和“Flags”字段的详细说明，可以参考/usr/include/linux/if_arp.h文件，也可以比照netstat命令的输出信息。

```
$ cat /proc/net/arp
IP address      HW type    Flags   HW address            Mask Device
169.254.78.56   0x1        0x2     00:0f:fe:22:29:d2    *      eth0
169.254.78.1    0x1        0x0     00:00:00:00:00:00    *      eth0
$
```

(2) /proc/net/dev文件

/proc/net/dev文件含有网络设备的状态统计信息，包括发送与接收的分组数据统计、错误与碰撞的统计数据，以及其他统计数据。dev文件的内容类似于ifconfig命令的输出结果，但ifconfig命令的输出信息可读性更强。例如：

```
$ cat /proc/net/dev
Inter-|   Receive                                          | Transmit
face |bytes  packets errs drop fifo frame compressed multicast|bytes  packets
errs drop fifo colls carrier compressed
lo:   5086    54    0    0    0    0    0    0    508654
0    0    0    0    0    0
eth0: 30961   348    0    0    0    0    0    0    30957    379
0    0    0    0    0    0
```

```

$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:40:b8:00:8e:52
          inet addr:169.254.78.100  Bcast:169.254.78.255  Mask:255.255.255.0
          inet6 addr: fe80::240:b8ff:fe00:8e52/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:363 errors:0 dropped:0 overruns:0 frame:0
          TX packets:389 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31861 (31.8 KB)  TX bytes:32707 (32.7 KB)
          Interrupt:11 Base address:0x2400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:54 errors:0 dropped:0 overruns:0 frame:0
          TX packets:54 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5086 (5.0 KB)  TX bytes:5086 (5.0 KB)

$

```

9./proc/stat文件

/proc/stat文件提供了自最近一次系统启动以来，系统内核收集的各种统计信息，其输出内容也可以通过vmstat和iostat等命令获得，只不过stat文件包含的是分类时间统计信息，而iostat（注意，运行iostat命令之前需要安装sysstat软件包）和vmstat命令给出的是每一类统计数据占整个时间的百分比。示例如下：

```

$ cat /proc/stat
cpu 36787 339 11530 547072 26699 291 110 0 0
cpu0 36787 339 11530 547072 26699 291 110 0 0
intr 1030952 592484 139 0 1 1 12 6 1 0 312135 1 1 109 0 42900 83141 0 0 0 0 0 0
.....
ctxt 2576801
btime 1258809076
processes 2508
procs_running 1
procs_blocked 0
softirq 663275 0 208811 148733 8350 50951 167079 0 542 7880
$ iostat
Linux 2.6.31-14-generic (iscas) 11/21/09 _i686_ (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           5.90    0.05    1.91    4.28    0.00   87.84

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
sda                  6.53         123.44         53.11     769319     331008
$ vmstat
procs -----memory----- --swap-- -----io----- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi   bo   in   cs us sy id wa
 1   0       0 347252 106008 356540    0    0    63   27  165  413  6  2 88  4
$

```

在`/proc/stat`文件的输出信息中，每个字段的意义简述如下：

- `cpu`——其中前7个数据分别是CPU处于用户模式、低优先级用户模式（`nice`）、系统模式、空闲状态、等待I/O完成、处理硬件中断及处理软件中断的时间（对于Intel x86系列计算机而言，时间计数单位为1/100秒）。其中第一行是所有CPU的累加值，随后依次给出每一个CPU的统计信息。上述例子表示当前系统仅配有一个CPU。
- `intr`——自系统启动以来处理中断的数量。其中第一列是中断处理的总和，随后的数字是针对每一类中断的分类统计。
- `ctxt`——系统经历的CPU上下文切换的次数。
- `btime`——系统启动的时间，其数值是从1970年1月1日算起的时间累加值（秒数）。
- `processes`——自启动以来系统创建的进程总和。
- `procs_running`——处于可运行状态的进程数量。
- `procs_blocked`——因等待I/O完成而处于等待状态的进程数量。

10./proc/swaps文件

`swaps`文件包含交换区存储空间的配置及其使用情况。其中包括每个交换区的文件名、交换区存储空间类型、交换区的总容量（KB）、当前占用的交换区容量（KB），以及交换区的优先级。优先级越低，交换区的选用越优先。如果系统中仅配备一个交换区，其输出信息示例如下：

```
$ cat /proc/swaps
Filename                                Type                                Size                                Used                                Priority
/dev/sda3                             partition                           843876                             16040                             -1
$
```

11./proc/sysvipc子目录

`/proc/sysvipc`子目录包含三个虚拟文件：`msg`、`sem`和`shm`。分别用于查询消息队列、信号灯和共享内存的运行状态信息。这些文件的内容类似于`ipcs`命令的输出结果。例如：

```
$ cat /proc/sysvipc/shm
      key      shmid      perms      size      cpid      lpid      nattch      uid      gid      cuid      cgid
      atime      dtime      ctime
      0      327680      1600      196608      1590      1656      2      1000      1000      1000      1000
1258844431 1258844431 1258844426
      0      360449      1600      393216      1626      1113      2      1000      1000      1000      1000
1258844427      0 1258844427
      0      229378      1600      196608      1538      1562      2      1000      1000      1000      1000
1258844414 1258844414 1258844412
.....
$
```

12./proc/uptime文件

`/proc/uptime`文件包含两个数值：系统自最近一次启动迄今为止的运行时间（秒）和系统的空闲时间（秒）。`uptime`文件第一个字段的值等同于`uptime`命令输出的第三个字段的值。只不过两者的时间单位及表示形式不同。例如：

```
$ cat /proc/uptime
17560.29 14692.03
```

```
$ uptime
01:58:44 up 7:06, 3 users, load average: 0.01, 0.03, 0.12
$
```

13./proc/vmstat文件

/proc/vmstat文件包含各种虚拟内存的统计信息，参见vmstat命令。

11.4 系统可调参数

/proc/sys是一个非常重要的目录，其中含有大量的文件，分别位于fs、kernel和net等子目录中，每个文件对应于系统内核的一个或多个可调参数变量。

利用kernel与net等目录中的文件，可以查询系统内核的可调参数值。作为超级用户，可以利用sysctl命令或sysctl()系统调用，修改系统内核的可调参数，也可以使用echo等命令把新的参数值直接写到相应的参数变量文件中（注意，不能使用sudo运行这里介绍的echo命令。为方便起见，可首先运行“sudo -i”命令，进入超级用户的Shell环境）。例如：

```
$ cat /proc/sys/net/ipv4/ip_forward
0
$ sudo -i
[sudo] password for gqxing:
# echo 1 > /proc/sys/net/ipv4/ip_forward
# cat /proc/sys/net/ipv4/ip_forward
1
#
```

但需要记住的是，这种内核参数的修改方式是临时性的，仅对当前的运行系统有效，一旦重新启动，系统将会重新恢复默认的内核参数设置。为了使定制的系统内核参数总是保持有效，详见最后一节的介绍。

11.4.1 文件系统可调参数

/proc/sys/fs目录主要包含与文件系统有关的系统内核参数。其中的file-max文件定义了系统范围内所有进程能够打开的文件数量，其默认值如下：

```
$ cat /proc/sys/fs/file-max
36056
$
```

如果在运行过程中经常出现文件描述符不足等错误信息，可以采用下列命令增加打开文件的数量限制（也可以使用setrlimit(2)系统调用设置每个进程能够打开的文件数量限制）：

```
# echo 100000 > /proc/sys/fs/file-max
#
```



设置file-max文件中的参数值时不能超过内核变量NR_OPEN规定的上限，参见/usr/include/linux/fs.h文件。

`/proc/sys/fs/file-nr`是一个只读文件，其中给出了三个重要的参数值：已分配的文件描述符的数量、空闲文件描述符的数量，以及文件描述符的最大数量。Linux系统内核采用动态的方式分配文件描述符，而且不会释放，因而无法回收再使用。如果已分配的文件描述符的数量接近于最大值，应当考虑增加最大值，例如：

```
$ cat /proc/sys/fs/file-nr
6848      0      36056
$
```

`/proc/sys/fs/inode-state`文件含有7个数值：`nr_inodes`、`nr_free_inodes`、`pre shrink`和4个空值（`/proc/sys/fs/inode-nr`文件只包含前两个参数值）。`nr_inodes`是系统已经分配的信息节点的数量（这个参数可能稍微大于`inode-max`，因为Linux系统分配信息节点采用的是一次分配一个整页面的方式）。`nr_free_inodes`是空闲信息节点的数量，例如：

```
$ cat /proc/sys/fs/inode-state
19410     5181      0          0          0          0          0
$
```

11.4.2 系统内核可调参数

`/proc/sys/kernel`是一个非常重要的子目录。其中的文件表示系统内核的可调参数。常见的或经常需要调整的内核参数包括`msgmax`、`msgmnb`、`msgmni`、`sem`、`shmall`、`shmmax`和`shmmni`等。

1. 消息队列

`/proc/sys/kernel/msgmax`文件定义了系统范围内单个System V消息的最大字节数。`/proc/sys/kernel/msgmni`文件定义了系统范围内消息队列标识符的最大数量限制。`/proc/sys/kernel/msgmnb`文件定义了系统范围内可以写入消息队列的最大字节数。下面的例子给出了系统的默认值：

```
$ cat /proc/sys/kernel/msgmax
8192
$ cat /proc/sys/kernel/msgmni
741
$ cat /proc/sys/kernel/msgmnb
16384
$
```

`/proc/sys/fs/mqueue`目录存在三个文件：`msg_max`、`msgsize_max`和`queues_max`，用于控制POSIX消息队列使用的资源。POSIX消息队列是一个全新的消息队列，与System V的消息队列完全不同，详见`mqueue-overview(7)`的说明。下面的例子给出了三个消息队列参数的默认值：

```
$ cat /proc/sys/fs/mqueue/msg_max
10
$ cat /proc/sys/fs/mqueue/msgsize_max
8192
$ cat /proc/sys/fs/mqueue/queues_max
256
$
```


2. 共享内存

/proc/sys/kernel/shmall文件包含系统范围内共享内存页面的数量限制。示例如下：

```
$ cat /proc/sys/kernel/shmall
2097152
$
```

/proc/sys/kernel/shmmax文件可用于查询和设置用户当前能够创建和使用的最大共享内存段的大小限制。Linux系统内核当前支持的最大共享内存段为1 GB。示例如下：

```
$ cat /proc/sys/kernel/shmmax
33554432
$
```

/proc/sys/kernel/shmmni文件用于指定系统范围内用户能够创建和使用的共享内存段的最大数量限制。示例如下：

```
$ cat /proc/sys/kernel/shmmni
4096
$
```

3. 信号灯

/proc/sys/kernel/sem文件包含4个IPC信号灯参数值：SEMMSL表示每个信号灯集合支持的最大信号灯数量；SEMMNS表示系统范围内所有信号灯集合能够支持的最大信号灯数量限制；SEMOPM表示每个semop()系统调用中能够指定的最大信号灯操作数量；SEMMNI表示系统范围内信号灯标识符的最大数量限制。示例如下：

```
$ cat /proc/sys/kernel/sem
250      32000   32      128
$
```

4. 网络可调参数

/proc/sys/net/ipv4目录存在的虚拟文件涉及各种网络设置，可用于查询和修改IP、TCP以及UDP等网络协议方面的可调参数。其中的许多设置可用于防止系统攻击，或设置系统的路由功能。

下面是/proc/sys/net/ipv4目录中的部分重要文件

- icmp_echo_ignore_all——表示系统内核是否忽略任何主机（包括自己）的ICMP ECHO分组数据。这个参数的默认值为0，意味着系统应响应ICMP ECHO分组数据。1表示忽略所有的ICMP ECHO请求。此时，任何主机都无法使用ping命令或其他类似的工具联系当前的主机系统。
- icmp_echo_ignore_broadcasts——除了区分是否发至广播或多播地址的ICMP分组数据外，其意义同上。变量的默认值为0。
- ip_default_ttl——用于设置默认的生存时间（Time To Live, TTL）参数，限制在到达目的主机之前，分组数据能够跳转的网络数量。ip_default_ttl的默认值为64。注意，增加这个参数值可能会降低系统的性能。每当经由一个路由器或网关时，分组数据的TTL值将减1。当TTL值减为0时仍未到达目的主机，分组数据将被扔掉。示例如下：

```
$ cat /proc/sys/net/ipv4/ip_default_ttl
64
$
```

- `ip_forward`——用于启用或禁用IP转发功能，即允许或禁止系统中的网络接口向另外一个网络接口转发分组数据。这个参数的默认值为0（即禁止转发分组数据）。如果把这个参数设置为1，表示允许转发网络分组数据。利用这个参数，可以启用或关闭在网络接口之间转发分组数据的功能，允许Linux系统作为一个防火墙或路由器。注意，这是一个极其重要的特殊变量，可用于设置NAT、防火墙、路由和伪装等。一旦改动这个参数值，能够导致其他配置参数的重置，使其恢复到默认状态。
- `ip_local_port_range`——用于指定TCP或UDP使用的端口范围。第一个数值是可用的最低端口号，第二个数值是可用的最高端口号。对于任何系统，当默认的端口号1024~4999仍不能满足要求时，可以利用这个参数，在32768~61000之间选定一个端口号范围。示例如下：

```
$ cat /proc/sys/net/ipv4/ip_local_port_range
32768 61000
$
```

- `tcp_keepalive_probes`——这个参数值告诉系统内核，在确认一个网络连接已经断开之前应发出多少keepalive试探分组数据。`tcp_keepalive_probes`的默认值为9，即在发出9个keepalive试探分组数据仍未收到回应时，可据此断定网络连接已经断开。示例如下：

```
$ cat /proc/sys/net/ipv4/tcp_keepalive_probes
9
$
```

- `tcp_keepalive_intvl`——用于指定链路空闲时仍然保持网络连接存活状态的时间长度（秒）。`tcp_keepalive_intvl`的默认值为75秒。这个参数值告诉系统内核，对于发出的每一个keepalive试探分组数据，能够容忍的等待回应时间有多长。这个参数值也用于计算网络连接的超时值。按照`tcp_keepalive_probes`的默认值为9计算，则默认的等待时间约为11分钟（75秒×9），超过此限，则断开连接。示例如下：

```
$ cat /proc/sys/net/ipv4/tcp_keepalive_intvl
75
$
```

- `tcp_keepalive_time`——这个参数告诉TCP/IP协议栈，如果网络连接一直空闲，应以何种时间频率（以秒为单位）发送TCP keepalive分组数据，以便保持网络连接的存活。`tcp_keepalive_time`变量的默认值为7200秒，也即2小时。如果需要调整，通常不应低于默认值，否则会造成不必要的网络资源消耗。示例如下：

```
$ cat /proc/sys/net/ipv4/tcp_keepalive_time
7200
$
```

5. 其他系统可调参数

`/proc/sys/kernel/ctrl-alt-del`文件中的参数值用于控制怎样处理控制台键盘的Ctrl-Alt-Del组合

键。当`ctrl-alt-del`文件中的参数值为0时，系统会随时检测Ctrl-Alt-Del组合键，一旦按下此组合键，将会把这一信号发送给init进程处理。init进程的默认处理方式是从容地重新启动系统。当这个参数值大于0时，将会立即重新启动系统。例如：

```
$ cat /proc/sys/kernel/ctrl-alt-del
0
$
```

`/proc/sys/kernel/domainname`和`/proc/sys/kernel/hostname`文件用于查询和设置当前系统的域名和主机名，其作用相当于执行`domainname`和`hostname`命令。例如，执行下面两个命令：

```
# echo "iscas" > /proc/sys/kernel/hostname
# echo "mydomain" > /proc/sys/kernel/domainname
```

相当于执行下面两个命令：

```
# hostname "iscas"
# domainname "mydomain"
```

`/proc/sys/kernel/panic`含有`panic_timeout`内核参数值。如果这个数值为0，在遇到致命的故障时，系统将会处于暂停状态。如果这个参数值为非0值，系统将会在遇到故障时暂停参数值指定的时间（秒），然后重新引导系统。默认的参数值如下：

```
$ cat /proc/sys/kernel/panic
0
$
```

`/proc/sys/kernel/panic_on_oops`文件用于控制系统遇到故障时的下一步处理动作。如果这个文件的参数值为0，系统将会尝试继续运行。如果参数值为1，系统将会在延迟数秒（以便`klogd`有时间记录错误信息）之后停机。如果`/proc/sys/kernel/panic`文件的参数值也是一个非0数值，将会重新启动系统。默认的参数值如下：

```
$ cat /proc/sys/kernel/panic_on_oops
0
$
```

`/proc/sys/kernel/pid_max`文件用于指定进程ID循环计数的最大值。在一个32位字长的系统中，这个参数的默认值是32768，进程的ID号不能超过这个参数值。当达到这个文件指定的数值时，系统将会从头开始重新分配进程的ID号。例如：

```
$ cat /proc/sys/kernel/pid_max
32768
$
```

`/proc/sys/kernel/printk`文件包含4个数值：`console_loglevel`、`default_message_loglevel`、`minimum_console_level`和`default_console_loglevel`。这4个参数值用于设定控制台错误信息输出或日志记录的级别。有关日志记录级别的说明详见`syslog(2)`。如果信息的优先级高于`console_loglevel`定义的级别，则相应的信息就能够输出到控制台。没有明显定义优先级的信息将按照`default_message_level`的设置处理。`minimum_console_loglevel`是`console_loglevel`能够设置的最小（最高）值。`default_console_loglevel`是`console_loglevel`的默认值。示例如下：

```
$ cat /proc/sys/kernel/printk
4      4      1      7
$
```

/proc/sys/kernel/pty目录存在两个文件，其中max文件定义了系统当前可以支持的最大伪终端设备数量。只读文件nr表示系统当前正在使用的伪终端设备数量。例如：

```
$ cat /proc/sys/kernel/pty/max
4096
$ cat /proc/sys/kernel/pty/nr
8
$
```

11.4.3 sysctl命令

在查询、修改系统可调参数时，除了能够使用前述的cat和echo命令之外，比较正规的方法是采用sysctl命令和sysctl.conf配置文件替代echo命令，修改/proc/sys目录中的任何文件，设置相应的系统参数。例如，可以使用下列sysctl命令：

```
$ sudo sysctl -w kernel.hostname=iscas
kernel.hostname = iscas
$
```

代替下列echo命令，设置系统的主机名：

```
# echo iscas > /proc/sys/kernel/hostname
#
```

sysctl命令可用于查询、动态地设置所有的系统内核参数，可以在系统运行期间观察、修改位于/proc/sys目录中的任何文件。sysctl命令的语法格式如下：

```
sysctl [-n] [-e] variable
sysctl [-n] [-e] [-q] -w variable=value
sysctl [-n] [-e] [-q] -p file
sysctl [-n] [-e] [-a | -A]
```

其中，“-n”选项表示仅输出系统可调参数的值；“-N”选项表示仅输出系统可调参数的参数名；“-e”选项表示忽略参数名字等出错信息；“-q”选项表示安静方式，禁止输出设置的系统可调参数值；“-a”选项表示输出系统内核的所有参数值；“-A”选项表示以表格形式输出系统内核的所有参数值。

variable是表示系统可调参数的变量，由/proc/sys子目录直至文件的完整路径名组成，但需要把路径名中的斜杠“/”换成句点“.”（实际上不替换也可以）。例如，为了查询当前的最大共享内存设置，可以使用下列命令：

```
$ sysctl kernel.shmmax
kernel.shmmax = 33554432
$
```

为了查询当前的TCP/IP设置是否支持路由功能，可以使用下列命令：

```
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
$
```

“-w variable=value”选项用于修改系统内核参数。其中，variable是系统内核参数的关键字，value是准备设置的值。如果value中包含引号或易被Shell解析的元字符，则需要使用双引号。注意，等号“=”前后不能有空格。例如，为了设置系统内核参数kernelhostname，修改系统的节点名，可以采用下列命令：

```
$ sudo sysctl -w kernel.hostname=iscas
kernel.hostname = iscas
$
```

如果想要了解系统提供哪些变量，能够用于设置系统内核的可调参数，可以访问/proc/sys目录，也可以使用“sysctl -a”或“sysctl -A”命令。例如：

```
$ sysctl -a
.....
kernel.hostname = iscas
kernel.domainname = <none>
kernel.shmmax = 33554432
kernel.shmall = 2097152
kernel.shmmni = 4096
.....
$
```

上面介绍的任何方法都是一种临时性的设置，仅对当前正在运行的系统有效，一旦关机后重新启动系统，系统将会恢复默认的设置，故仅能用于测试的目的。为了一劳永逸地设定系统的可调参数，可以利用sysctl命令的“-p file”选项，借助/etc/sysctl.conf文件实现。“-p file”选项表示从指定的文件（默认的文件为/etc/sysctl.conf）中读入并设置其中列出的所有系统参数。

每次引导系统时，init进程都会按照/etc/init/procps.conf作业配置文件的定义，调度执行其中指定的任务。procps.conf作业配置文件包含一个sysctl命令，采用/etc/sysctl.conf等文件作为参数，设置系统的可调参数，其代码如下（有关作业配置文件的说明参见第15章“系统启动与关机”）：

```
$ cat /etc/init/procps.conf
.....
description      "set sysctls from /etc/sysctl.conf"

start on virtual-fileSystems

task
script
    cat /etc/sysctl.d/*.conf /etc/sysctl.conf | sysctl -p -
end script
$
```

因此，加到/etc/sysctl.conf文件中的任何参数设置将会在系统的启动过程中开始生效。如果需要改变系统内核参数的默认设置，如增加系统的共享内存，使系统支持路由功能等，可以把有关的参数设置加到/etc/sysctl.conf文件中，以便在系统的启动过程中设定系统的可调参数。下面是系统提供的/etc/sysctl.conf文件的部分内容：

```
$ cat /etc/sysctl.conf
.....
#kernel.domainname = example.com
```

```
# Uncomment the following to stop low-level messages on console
#kernel.printk = 4 4 1 7
.....
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#net.ipv6.conf.all.forwarding=1
.....
$
```

第12章 磁盘空间管理

本章主要介绍磁盘空间管理，说明如何查询磁盘空间的使用与空闲情况，怎样找出超大容量的文件，以及长期闲置不用的文件，定期清除磁盘中的垃圾文件，利用Linux系统提供的各种工具备份或恢复文件，配置文件系统磁盘空间和信息节点的使用限额等。

12.1 查询磁盘空间信息

磁盘空间管理的主要目的是了解磁盘存储空间的使用情况，包括系统当前已经使用的空间、可用的空闲空间、现有的文件数量、空闲的信息节点等；及时清理垃圾文件（如内存影像转储文件），删除超大容量和长期闲置不用的文件，以及清除临时目录或文件。在此基础上，可以利用Linux系统提供的标准工具，复制、备份或恢复文件甚至整个文件系统；设置磁盘空间的限额，确保磁盘空间资源的合理分配与使用等。

当文件系统空间容量的使用接近90%时，需要利用cp等命令，把其中的文件转储到相对空闲的其他磁盘中，或利用tar、cpio、dd以及dump等命令把文件转储到磁带上，或者干脆删除其中无需继续保存的文件。

12.1.1 常用磁盘空间管理工具

表12-1 给出了部分磁盘空间管理的常用工具。

表12-1 磁盘空间管理的常用工具

命令	简单说明
df	查询文件系统中的可用或已用存储空间及文件信息节点数量
du	查询指定（或当前）目录中每个文件或目录占用的磁盘空间
find -size	检索指定目录中指定大小的文件
ls -s	以1 KB数据块为单位，显示文件的大小
cpio	用于创建、转储或恢复cpio档案文件，实现文件或文件系统的备份与恢复。也可用于实现整体目录层次结构的复制
tar	用于创建、转储或恢复tar档案文件，实现文件或文件系统的备份与恢复
dd	用于实现原始数据复制。可以复制文件甚至文件系统（也即整个磁盘分区）

12.1.2 使用df命令查询空间使用情况

系统管理员经常需要监控磁盘空间的使用情况。即使系统配置的硬盘比较大，如果分区不当，如“/”文件系统过小，仍然会产生磁盘空间紧张的情况。为了监控磁盘空间的使用情况，自然会用到df命令。利用df命令，可以查询每个文件系统磁盘空间的使用与空闲状况。df命令的语法格式简写如下：

```
df [-ahikltTv] [-B size] [-t fstype] [-x fstype] [fileys]
```

表12-2 给出了df命令的部分常用选项和参数及其简单说明。

表12-2 df命令的部分常用选项和参数

选项与参数	GNU选项	简单说明
-a	--all	显示所有文件系统（包括虚拟文件系统，如/proc）的存储空间及其使用情况
-B size	--block-size=size	以指定的字节数量为单位，显示每个已安装文件系统的磁盘空间使用情况。输出信息包括文件系统的设备文件名、文件系统总容量、已分配的存储空间容量、可用的存储空间容量、已用存储空间占文件系统总容量的百分比，以及文件系统的安装点。其中，size可以是一个单独的数字，也可以附加一个字符K、M或G等，如1K、1M或1G，分别表示以1KB、1MB或1GB为单位
-h	--human-readable	以KB、MB或GB为单位，显示每个已安装文件系统的存储空间使用情况。输出信息包括文件系统的设备文件名、文件系统总容量、已分配的存储空间容量、可用的存储空间容量、已用存储空间容量占文件系统总容量的百分比，以及文件系统的安装点
-i	--inode	显示文件系统的设备文件名、文件系统的信息节点（文件）总量、空闲信息节点数量、已用信息节点数量、已用信息节点数量占信息节点总量的百分比，以及文件系统的安装点
-k		以KB为单位，显示每个文件系统的存储空间使用情况。输出信息包括文件系统的设备文件名、文件系统的总容量、已分配的存储空间容量、可用的存储空间容量、已用存储空间占文件系统总容量的百分比，以及文件系统的安装点。在Ubuntu Linux系统中，“-k”是df命令的默认选项
-l	--local	显示已安装的本地文件系统的存储空间使用情况，包括可用的存储空间容量，以及可用的文件信息节点数量等
-t fstype	--type=fstype	显示指定文件系统类型的磁盘空间总量与可用容量，信息节点（文件）总量与可用信息节点数量
-T	--print-type	同时输出每个文件系统的类型
-x fstype	--exclude-type=fstype	显示除指定文件系统类型之外的其他文件系统的磁盘空间总量与可用容量、信息节点（文件）总量与可用信息节点数量
fileys		指定文件系统、磁盘分区的设备文件名或文件系统的安装点。通常，df命令仅显示本地系统已经安装的所有文件系统的存储空间使用信息

如果不加任何选项和参数，df命令通常会以KB为单位，显示系统中所有已经安装的文件系统的存储空间使用情况，包括可用数据块数量。下列df命令的输出数据表明，“/”文件系统中尚有5584908个可用的数据块，也即还有约5.5GB的可用存储空间：

```
$ df
Filesystem      1K-blocks    Used   Available Use% Mounted on
/dev/sda2        14128244    3315076    10095488  25% /
.....
/dev/sda1        13285720    9415112     3870608  71% /media/A22eXP
$
```


表12-3 给出了df命令输出信息中各个字段的简单说明。

表12-3 df命令输出字段的说明

字段名	简单说明
1 K-blocks、1 M-blocks或1 G-blocks	文件系统中存储空间的总容量
Used	文件系统中已经占用的存储空间数量
Available或Avail	文件系统中可用的空闲存储空间数量
Use %	文件系统中已用存储空间数量占全部数据存储空间总量的百分比
Size	文件系统中全部存储空间的总容量（参见“df -h”命令的输出）
Mounted on	安装点

如果df命令的输出信息表明文件系统的使用已经接近或超过整个容量的90%，则需要清除不必要的文件，删除临时文件或长期闲置不用的文件，或者使用后面将要介绍的各种系统工具，复制或备份部分文件，以增加可用的存储空间。

在较大的文件系统中，以KB为固定度量单位计数存储空间，尚需经过一番换算才能准确地知道输出数值的意义，故不太符合人的直观思维与阅读习惯。为此，可以使用“df -h”命令，以更容易阅读的形式，即以KB、MB或GB等为计数单位，显示文件系统的相关信息，包括每个文件系统的总容量、当前已经使用了多少存储空间、尚有多少可用的存储空间、实际使用的空间占整个文件系统总容量的百分比，以及文件系统的安装点等信息。例如：

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2       14G   3.2G   9.7G  25% /
.....
/dev/sda1       13G   9.0G   3.7G  71% /media/A22eXP
$
```

上述df命令仅仅给出了文件系统的存储空间及其使用情况。为了获取文件系统的信息节点及其使用情况，可以使用带有“-i”选项的df命令。

```
$ df -i
Filesystem      Inodes   IUsed   IFree IUse% Mounted on
/dev/sda2       897600   157988  739612   18% /
.....
/dev/sda1       3903376   19852  3883524    1% /media/A22eXP
$
```

在上述输出信息中，Inodes字段表示文件系统的信息节点总量，IUsed字段表示已用信息节点的数量，IFree字段表示空闲信息节点的数量，IUse%字段表示已用信息节点数量占信息节点总量的百分比。

从上述例子中可见，df命令的输出数据中还包含部分虚拟文件系统信息，如udev等。对于了解存储空间的实际使用情况，此类信息是无关紧要的，我们真正关心的是实际的文件系统。因此，为了避免输出其他干扰信息，可以使用“-t”选项，指定文件系统的类型，从而限定df命令的输出结果：

```
$ df -t ext4
Filesystem      1K-blocks      Used Available  Use% Mounted on
/dev/sda2        14128244    3315028   10095536   25% /
$
```

如果不知道每个已安装文件系统的类型，可以查阅/etc/fstab 文件，或使用df命令的“-T”选项:

```
$ df -T
Filesystem      Type      1K-blocks      Used  Available Use% Mounted on
/dev/sda2      ext4      14128244    3315060   10095504   25% /
.....
/dev/sda1  fuseblk   13285720    9415112   3870608    71% /media/A22eXP
$
```

12.1.3 使用du命令查询已用存储空间

du命令用于显示指定目录（或当前目录）中每个子目录或文件占用的磁盘空间数量。如果不加任何选项和参数，du命令通常会输出以1 KB数据块为单位的统计数据。du命令的语法格式简写如下:

```
du [-abchkms] [-B size] [directory]
```

表12-4 给出了du命令的部分常用选项和参数及简单说明。

表12-4 du命令的部分常用选项和参数

选项与参数	GNU选项	简单说明
-a	--all	以KB为单位，列出指定目录或当前目录中每个文件、每个子目录，以及其中每个文件占用的磁盘空间数量，最终给出整个目录占用的磁盘空间总量。如果指定的参数是一个普通文件，则显示指定文件占用的存储空间
-B size	--block-size=size	以指定的字节数为计数单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量。其中，size可以是一个单独的数字，也可以附加一个K、M或G等字符，如1 K、1 M或1 G，分别表示以KB、MB或GB为单位
-b	--bytes	等价于指定了“--block-size=1”选项
-c	--total	以KB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量，最后给出累计占用的磁盘空间总量
-h	--human-readable	以KB、MB或GB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量
-k	--kilobytes	以KB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量。在Ubuntu Linux系统中，“-k”是du命令的默认选项
-m	--megabytes	以MB为单位，显示指定目录或当前目录，以及其中所有子目录占用的磁盘空间数量
-s	--summarize	以KB为单位，显示指定目录或当前目录（包括其中的所有子目录和文件）占用的磁盘空间数量
directory		指定准备检查的目录。如果同时指定多个目录，目录之间需加空格分隔符

利用du命令，当发现较大的目录或文件时，可以视具体情况，确定是否保留、删除或异地备份，以节省磁盘的存储空间。下面是一个du命令输出的例子：

```
$ du /home/gqxing/data
2588    /home/gqxing/data/backup
3816    /home/gqxing/data/security
1632    /home/gqxing/data/recent
8092    /home/gqxing/data
$
```

同样，为了避免按数据块形式列出统计数据，可以使用“-h”选项，以1 KB、1 MB或1 GB等符合阅读习惯的形式输出每个目录或文件占用的存储空间统计数据。下面的例子说明怎样利用“-h”选项，以适当的单位输出存储空间的使用情况：

```
$ du -h /home/gqxing/data
2.6M    /home/gqxing/data/backup
3.8M    /home/gqxing/data/security
1.6M    /home/gqxing/data/recent
8.0M    /home/gqxing/data
$
```

当输出的文件比较多，且文件的大小参差不齐时，可以采用管道机制，利用sort命令进行排序。其中，“-r”选项表示反向排序，按从大到小的顺序列出每一个目录和文件占用的磁盘空间数量。“-n”选项表示按数值而非字符顺序排序。例如：

```
$ du -k /home/gqxing/data | sort -rn
8092    /home/gqxing/data
3816    /home/gqxing/data/security
2588    /home/gqxing/data/backup
1632    /home/gqxing/data/recent
$
```

如果只想查询某个目录（包括其中的所有文件和子目录）占用的全部存储空间数量，可以使用带有“-s”选项的du命令。下面的例子仅仅统计了/boot目录占用的存储空间数量：

```
$ du -s /boot
43740   /boot
$
```

12.1.4 使用find命令找出超大文件

当存储空间紧张，需要尽快腾出磁盘空间时，删除超大容量的文件或把文件备份到其他存储介质上是一种快速有效的方法。为了列出超过指定大小的文件，可以使用find命令，其语法格式简写如下：

```
find directory -size +nnn -print
```

其中，directory是起始检索目录，“-size +nnn”选项表示大于指定数量的数据块（512个字节）的数量，“-print”选项表示输出find命令的检索结果。这个命令将会列出超过指定大小的所有文件。

例如，下面的例子说明怎样利用find命令，从指定的/usr/share/scim目录中找出大小超过2048个数据块（1MB）的所有文件。

```
$ find /home/gqxing/data -size +2048 -print
/home/gqxing/data/backup/config
/home/gqxing/data/security/specifications
/home/gqxing/data/recent/transactions
$
```

上述命令的输出结果仅仅给出了文件的名字，这些文件究竟有多大并不知道。同样，为了便于观察，最好能够按文件的大小输出最终的结果。为此，可以使用管道，把find、ls、gawk和sort等命令组合起来，按从大到小的顺序显示一个文件列表。例如：

```
$ cd /home/gqxing/data
$ find . -size +2048 -exec ls -ls {} \+ | gawk '{print $1 "\t" $10}' | sort -rn
2932    ./security/specifications
2572    ./backup/config
1628    ./recent/transactions
$
```

12.1.5 使用find命令找出闲置文件

作为系统管理员，日常系统维护的一个主要工作就是清除系统中长期闲置不用的文件或垃圾文件，清除临时文件或目录。为了找出在指定的时间内一直没有使用过的文件，可以使用下列find命令：

```
find directory -type f [-atime +nnn] [-mtime +nnn] -print
```

其中，directory表示准备检索的起始目录，“-atime +nnn”选项用于找出指定天数（nnn）内没有访问过的文件，“-mtime +nnn”选项用于找出指定天数（nnn）内没有改动过的文件。



取决于指定的起始检索目录，find命令的输出结果可能包含一系列系统文件或用途不明的文件。而系统文件即使长时间未用也是不应轻易删除的，除非自己有绝对把握。在不确定其用途时，不能随意删除系统文件。即使确实没有用处，也不要直接删除，最好能够利用下列dpkg命令，确定文件所属的软件包，然后再利用软件包管理工具删除不必要的软件包：

```
$ dpkg -S filename
```

因此，最保险的方法是先利用find命令，找出指定时间内一直没有使用过的文件，把检索结果储存在一个临时文件（如filelist）中，确定是否应当删除。如果检索出来的文件均属无用的文件（或删除其中需要保留的文件后剩下的全部为无用文件），再使用下列命令删除检索出来的文件：

```
$ rm 'cat filelist'
```

下面的例子展示了怎样找出/home/guest目录及其子目录中两个月来一直没有访问过的文件，并把这些文件列表存入/tmp/filelist文件中，经过确认之后，再使用rm命令予以删除。

```
$ cd /home/guest
$ find . -type f -atime +60 -print > /tmp/filelist
$ cat /tmp/filelist
./doc/TechnicalSpecifications.doc
./doc/TroubleShootingGuide.doc
./Config/NetworkSetup.doc
./Config/PrinterSetup.doc
$ rm 'cat /tmp/filelist'
$
```

12.1.6 使用find命令处置core文件

在开发和测试期间，由于程序可能存在这样或那样的问题，开发系统中经常会存在大量的内存映像文件，也即core文件。这样的文件一多，既影响目录文件的清洁性，又会占用大量宝贵的存储空间。为此，作为系统管理员或开发者本人，应当经常清理系统中或开发者自己的工作目录中存在的core文件。

为了找出并删除这种core垃圾文件，可以使用下列find命令，在指定的目录或当前工作目录中，检索并删除core文件。

```
find directory -name core -exec rm {} \;
```

其中，directory是起始检索目录，“-name core”选项表示检索名为core的内存映像文件，“-exec rm {} \;”选项表示执行rm命令，删除检索出来的core文件。

下面的例子展示了怎样使用find命令找出并删除hwang用户主目录中的core文件。

```
$ cd /home/hwang
$ find . -name core -exec rm {} \;
$
```

12.1.7 使用ls命令检测文件的大小

前面曾经介绍过，ls命令的“-s”选项可用于显示以1 KB为计数单位的文件大小，例如：

```
$ cd /home/gqxing/doc
$ ls -ls
总计 2180
464 -rw-r--r-- 1 gqxing gqxing 471040 2009-07-04 18:10 design.doc
460 -rw-r--r-- 1 gqxing gqxing 465920 2009-08-16 15:20 project.doc
316 -rw-r--r-- 1 gqxing gqxing 317440 2009-06-08 09:40 proposal.doc
416 -rw-r--r-- 1 gqxing gqxing 419840 2009-10-21 16:50 report.doc
412 -rw-r--r-- 1 gqxing gqxing 414720 2009-06-20 11:48 requirement.doc
112 -rw-r--r-- 1 gqxing gqxing 110240 2009-09-26 10:42 testing.doc
$
```

为了更便于观察，还可以组合使用sort命令，按照从大到小的顺序列出每个文件，例如：

```
$ cd /home/gqxing/doc
$ ls -s | sort -rn
464 design.doc
460 project.doc
```

```
416 report.doc
412 requirement.doc
316 proposal.doc
112 testing.doc
0<^Δ 2180
$
```

12.2 采用标准工具备份与恢复数据

备份与恢复是两个互逆的数据处理过程，如果运用得当，能够防止系统丢失重要的数据。备份通常指的是从系统磁盘中把系统数据和业务数据等复制到另一个存储介质（通常为移动介质，如磁带、硬盘、软盘或CD/DVD等）的过程。恢复是一个逆过程，即从备份介质中把重要文件、文件系统或业务数据复制到系统中。

此外，还要考虑备份什么数据以及怎样备份，这涉及到备份的策略，也影响到数据的恢复。在Linux系统中，通常可以备份单个文件、目录、文件系统或数据分区，这取决于系统中数据修改的频繁程度、文件的重要性以及其他因素。备份的频率可以是每天一次、每周一次，甚至每月一次等。

从备份的数据考虑，可以采用下列备份方式之一：

- 文件备份——采用这种备份方式，可以备份文件系统中的重要系统文件和目录等。一旦系统文件出现问题，可以直接恢复单个或部分文件。
- 文件系统备份——采用这种备份方式，可以备份从根目录开始的整个文件系统，也可以备份/var、/home或用户自建的单个文件系统等。当文件系统出现问题时，可以恢复整个或单个文件系统。
- 数据分区备份——在某些数据库应用系统中，其数据通常采用原始分区而非文件系统存储数据。针对这种情况，可以采用数据分区的备份方式。当需要恢复这样的备份数据时，必须恢复整个磁盘分区。

从备份的方式考虑，可以采用下列备份方式之一：

- 完整备份——这种备份方式主要用于备份文件系统。利用这种备份方式，能够完整地恢复一个文件系统。
- 增量备份——增量备份是在完整备份或其他增量备份的基础上，仅仅备份自上次备份以来发生变动的文件。增量备份用于补充相应时间周期内的完整备份。在一个特定的时间周期内，数据的备份应当包括一个完整备份与0个或多个增量备份。

在备份与恢复数据文件或文件系统时，可以直接使用Linux系统提供的标准工具实现，如cpio、tar或dd等。也可以采用Linux系统的专用工具，如dump/restore、amanda、BackupPC、rsync或bacula等。

cpio、tar和dd等命令是最基本的数据备份与恢复工具，不仅适用于Linux系统，而且广泛应用于各种UNIX系统。若想备份单个文件、部分文件或整个文件系统，完全可以使用Linux系统的标准工具实现。因此，本节仅讨论Linux系统提供的标准工具，下一节再介绍专用的备份与恢复工具。至于备份使用的介质，则不外乎磁带、磁盘或光盘等。

在选择备份工具时,也可以考虑各种应用软件提供的备份工具,如MySQL数据库的mysqldump或mysqlhotcopy等命令。

为了备份一组文件、一个完整的目录或文件系统,如/home/gqxing/data目录,通常应使用du和df等命令,事先确定目录或文件系统的容量,以便能够确定生成的档案文件究竟有多大。如果需要把生成的档案文件写入外部的存储介质中,也可以据此选择存储介质的类型、规格和数量。示例如下:

```
$ cd /home/gqxing/data
$ du -s .
1235224 .
$
```

du命令的输出数据表示,上述目录中的文件总共占用了1 235 224个数据块(1 KB)的存储空间。如果想要备份这些文件,至少需要1.2 GB的存储空间。

12.2.1 利用cpio命令实现数据备份与恢复

cpio是一个常用的数据备份与恢复工具,能够复制单个文件、一组文件、一个完整的目录结构,甚至一个完整的文件系统,把选定的文件复制为一个档案文件,直接存储于磁带、磁盘或其他存储介质中。例如,cpio命令能够把一个磁盘分区中的文件系统完整地复制到另一个磁盘分区。

在复制过程中,cpio命令将会在每个文件之间(之前)插入一个头信息,以便能够恢复任何选定的文件。cpio命令也能够识别存储介质的结束标志,因而能够自动提示用户更换新的介质,根据备份文件的存储容量要求,把档案文件存储到多个介质中。当需要使用多个介质(如多个磁带)创建档案文件,备份文件或文件系统时,cpio是一个比tar等更灵活、更有效的工具。

cpio的工作原理是,首先利用find等命令生成一个文件列表,然后通过管道提交给cpio命令,cpio再把给定的文件,包括其属性信息,复制到指定的档案文件或存储介质中。

因此,在使用cpio命令时,可能需要频繁地使用find等命令,为cpio命令提供需要复制的文件或文件列表,并通过管道送交cpio命令。

cpio具有如下三种运行模式,或者说具有三种用途:

- 恢复: 利用“-i”选项,把先前创建的档案文件恢复到系统中。
- 备份: 利用“-o”选项和find命令等工具,从标准输入中读取文件列表,创建一个档案文件,把选定的文件,包括路径名及文件属性等信息写入其中,以便能够把文件原样恢复到系统中。
- 复制: 利用“-p”选项和find命令等工具,从标准输入中读取文件列表,把选定目录或文件系统中的文件原样复制到另外一个目录位置或文件系统中,从而创建一个完整的目录或文件系统副本。

cpio命令的语法格式简写如下。表12-5给出了cpio命令的常用选项及简单说明。

```
cpio -i [-cdfmtuvV] [-C bufsize] [-F file] [-I file [-M message]] [pattern]
      [< file]
cpio -o [-aABcvV] [-C bufsize] [-F file] [-O file [-M message]] [> file]
cpio -p [-admuvV] directory
```


表12-5 cpio命令的常用选项

选项	GNU选项	简单说明
-i	--extract	输入模式，用于恢复档案文件
-o	--create	输出模式，用于备份档案文件
-p	--pass-through	复制模式，用于复制一个完整的目录或文件系统
-a	--reset-access-time	在复制完成之后，复原输入文件的访问时间，使输入文件仍然保持原来的访问时间，就像cpio命令根本没有读过这些文件一样
-A	--append	把新复制的文件附加到档案文件的后面。这个选项仅适用于输出模式。在与“-C”或“-F”选项一起使用时，指定的档案文件必须是一个磁盘文件
-B		以5120个字节作为数据块的读写单位。默认的数据块缓冲区为512个字节。注意，输入和输出时最好采用相同的数据块参数
	--block-size= <i>n</i>	以“ <i>n</i> × 512”个字节作为数据块的读写单位
-c		以ASCII字符格式读写头信息，确保不同的Linux系统或UNIX系统之间的兼容性
-C <i>size</i>	--io-size= <i>size</i>	以指定的字节数作为数据块的读写单位
-d	--make-directories	在复制过程中，根据需要创建必要的目录。注意，这个选项只能与“-i”或“-p”选项一起使用
-f	--nonmatching	仅复制与指定模式不匹配的文件
-F <i>file</i>	--file= <i>file</i>	使用指定的档案文件代替标准输入或标准输出
-I <i>file</i>		使用指定的档案文件作为输入数据，而不是读取标准输入。如果指定的文件是一个字符特殊设备文件，且当前介质已经完全读入，可根据提示更换新的介质，然后按下“Enter”键，以便继续读取下一个介质中的数据。注意，这个选项只能与“-i”选项一起使用
-m	--preserve-modification-time	使复制的文件仍然保持先前的修改时间。注意，这个选项在复制目录时无效
-M <i>message</i>	--message= <i>message</i>	定义一个提示信息。当读写达到存储介质结尾，需要更换新的介质时，提示用户更换下一个介质。如果提示信息中包含字符串“%d”，系统将代之以当前的介质序号（介质从1开始编号）
-C <i>file</i>		使用指定的档案文件作为输出文件，而不是写到标准输出。如果指定的文件是一个字符特殊设备文件，且当前介质已经完全写满，可根据提示更换新的介质，然后按下“Enter”键，以便继续写入下一个介质。注意，这个选项只能与“-o”选项一起使用
-t	--list	读取并显示档案文件中的文件列表
-u	--unconditional	无条件地强行复制。如果不加此选项，cpio命令的常规处理惯例是禁止同名的老文件替换或覆盖新文件
-v	--verbose	显示方式。输出文件名列表及其相关属性。当与“-t”选项一起使用时，其效果如同“ls -l”命令输出的文件名列表
-V	--dot	特殊显示方式。对于读写的每一个文件，仅仅输出一个句点“.”标记

假定想要备份/home/gqxing/data目录中的所有文件，把生成的档案文件写入磁盘中，可以使用下列cpio命令：

```
$ cd /home/gqxing/data
$ find . -print | cpio -oc > /backup/data.cpio
1024050 blocks
$
```

或

```
$ find /home/gqxing/data -print | cpio -oc > /backup/data.cpio
1024050 blocks
$
```

在上述例子中，find命令用于提供cpio命令需要复制的文件。由于“find . -print”命令使用当前目录“.”作为起始检索路径，其生成的文件列表为相对路径名，故前者将会采用相对路径名的形式，把所有文件备份到一个指定的data.cpio档案文件中。而第二个find命令采用的是绝对路径名，其生成的文件列表也为相对路径名，故后者生成的是一个包含绝对路径名的档案文件。

采用相对路径复制档案文件的最大好处是，在恢复档案文件时能够重新定位，抽取的文件可以集中存储在任何目录位置，不必把文件仍然恢复到原来的目录位置，故而能够避免覆盖原有的文件。如果先前复制的档案文件采用的是相对路径名，当使用cpio命令恢复档案文件时，读入的文件将会存放到当前目录中。因此，在使用cpio命令复制档案文件时，通常应事先进入指定的目录，然后在当前目录下执行find和cpio命令，使生成的档案文件不包含绝对路径名。

但是，如果复制档案文件时采用的是绝对路径名，在恢复档案文件时也会采用相同的绝对路径名读入文件，覆盖原有的目录和文件，而无法在恢复文件时重新定位，因而也就无法改变文件的存储位置，只能按照原来的目录结构原样恢复。由此可见，使用绝对路径名的文件备份方式具有潜在的危险性。如果现有的文件有所变动，恢复文件后将会使改动的文件丢失数据（当然，有时也许恰恰需要这样做——覆盖修改有误的文件）。

对于采用了相对路径名创建的cpio档案文件，如果想把整个档案文件全部恢复到一个新的目录中，可以事先进入目的目录，然后使用下列cpio命令：

```
$ cd /newdir
$ cpio -icdmu < /backup/data.cpio
1024050 blocks
$
```

其中，“-i”选项表示以输入模式执行cpio命令，从输入的档案文件中抽取文件。“-c”选项表示读取ASCII字符格式的文件头信息。“-d”表示必要时可以在复制过程中创建相应的目录。“-m”选项表示复制的文件仍保持原先的访问时间。“-u”选项表示无条件地强行复制文件，即使存在同名的新文件，不管新旧与否，均强行覆盖，否则老的文件无法替代新文件。在原封不动地复制一个目录或文件系统时，这个选项尤其有用，因为同一文件之前可能已经存在于目的目录或文件系统中。

在完成文件的复制之后，cpio将会输出其读写的数据块（512个字节）数量，表示生成或读取的档案文件的大小。

除了备份与恢复功能之外，利用“-p”选项的复制功能，cpio命令还能够复制整个目录或文件系统。为了复制一个完整的目录或文件系统，把其中的文件全部复制到另外一个目录或磁盘分区（文件系统）中，应首先使用cd命令进入源目录或文件系统的根目录，然后再利用find和cpio命令实现整个目录或文件系统的复制，例如：

```
$ cd dir1
$ find . -depth -print | cpio -pdmu dir2
1024048 blocks
$
```

其中，find命令的“-depth”选项表示逐层深入各级子目录，自底向上依次检索所有的文件，“-print”选项表示输出检索出来的文件名。cpio命令的“-p”选项表示采用复制模式，复制一个完整的目录。

执行上述命令之后，cpio将会把dir1目录的文件，按照原有的目录层次结构，完全复制到新的目录dir2中。进入dir2目录之后，即可看到新复制的目录或文件系统与原来完全一样。

下面以磁带（其设备文件名为/dev/st0）和磁盘档案文件为例，说明怎样利用cpio和find命令备份或恢复文件、目录，甚至整个文件系统。实际上，这里介绍的备份与恢复过程同样也适用于其他存储介质，如磁盘分区或CD/DVD等，只需把输入或输出文件名改换为相应的设备文件名即可。

1. 使用cpio命令把文件复制到磁带上

为了把某个目录下的所有文件复制到一个磁带上，可以采用find等命令提供需要复制的文件列表，然后通过管道提交给cpio命令，再由cpio把文件复制到磁带上。例如，假定希望备份/home/gqxing/data目录中的所有数据文件，可首先进入该目录，然后执行下列cpio命令：

```
$ cd /home/gqxing/data
$ find . -print | cpio -ocvB > /dev/st0
.
./data.0630
./data.0731
./data.0831
./data.0930
./data.1031
102405 blocks
$
```

在上述例子中，find命令用于提供cpio命令需要复制的文件。cpio命令中的“-o”选项表示以备份方式执行cpio命令，创建一个档案文件。“-c”选项表示采用ASCII字符格式写入头信息，以便确保新建档案文件的兼容性。“-v”选项表示在屏幕（标准输出）上显示复制的文件名。

“-B”选项表示以5120个字节的数据块为读写单位。“>/dev/st0”表示指定的输出文件为磁带机。

执行上述命令之后，当前目录中的所有文件将会归档并复制到磁带上。如果磁带中有数据，原有的数据将会被覆盖。命令输出的最后一行数据是复制的数据块（512字节）数量，表示生成的档案文件的大小。在这个例子中，生成的档案文件约为512MB。

为了验证数据是否确实已经复制到磁带上，可以使用下列命令：

```
$ cpio -ict < /dev/st0
.
data.0630
data.0731
data.0831
data.0930
data.1031
1024050 blocks
$
```

其中，“-t”选项意味着仅仅读取并显示档案文件中的文件列表。“</dev/st0”表示指定的输入文件为磁带机。输出结果表示，读取的文件内容与上述档案文件制作过程的输出结果完全一致。

2. 使用cpio命令把文件复制到多个磁带上

如前所述，当使用cpio命令，把单个文件、一组文件或整个文件系统复制到磁带上时，应注意整个数据的容量。如果数据量过大，一个磁带无法容纳所有的数据，需要使用多个磁带介质时，应注意使用“-I”选项，以便在磁带复制完成之后，由系统提示用户更换一个新的磁带，从而把档案文件备份到多个磁带上。例如：

```
$ cd /home/gqxing/data
$ find . -depth -print | cpio -ocvB -M "The tape has reached end. Please replace
it with #2. Press Enter when ready." -O /dev/st0
```

当复制到中途时，系统会输出下列提示信息，提醒用户更换新的磁带：

```
The tape has reached end. Please replace it with #2. Press Enter when ready.
```

更换新的磁带后，按下Enter键，系统将会继续复制文件，直至把所有的文件均写入磁带中。

在上述例子中，“-M”选项用于指定更换介质时输出的提示信息；“-O/dev/st0”选项表示指定的输出文件为磁带机。

3. 使用cpio命令恢复档案文件中的所有文件

由于先前在复制磁带档案文件时采用了相对路径名，当使用cpio命令读取磁带档案文件时，读入的文件将会存放到当前工作目录中。因此，在恢复文件之前可以任选一个目录。

例如，为了把先前备份到磁带上的档案文件恢复到系统中，首先选择一个目录，然后执行下列cpio命令（其中，“-v”选项表示在屏幕上显示抽取的文件），把先前复制的所有文件恢复到选定的目录中：

```
$ cd /newdir
$ sudo cpio -icduvB < /dev/st0
.
data.0630
data.0731
data.0831
data.0930
data.1031
102405 blocks
$
```

4. 使用cpio命令恢复档案文件中的指定文件

上述cpio命令把档案文件中的文件全部恢复到系统中。事实上，也可以从复制的档案文件中抽取单个文件或一组文件。为此，可以使用下列cpio命令：

```
$ cpio -icduv "pattern" < /dev/st0
```

其中，“pattern”表示从档案文件中抽取匹配指定模式的所有文件，并恢复到当前工作目录中（注意，如果需要同同时指定多个模式，每个模式必须加双引号）。下面的例子说明了怎样抽取文件名后缀为“0831”的文件：

```
$ cd /newdir
$ sudo cpio -icduv "**0831" < /dev/st0
data.0831
1024050 blocks
$ ls -l *0831
-rw-r--r-- 1 gqxing gqxing 104861696 2009-08-31 23:30 data.0831
$
```

5. 使用cpio命令查询先前备份的档案文件

为了查询先前备份的档案文件，可以使用下列cpio命令（其中，“-t”选项表示仅仅读取并显示档案文件中的文件列表）：

```
$ sudo cpio -ict < /dev/st0
.
data.0630
data.0731
data.0831
data.0930
data.1031
1024050 blocks
$
```

如果使用“-v”选项，cpio命令还会给出更多的文件属性信息。例如：

```
$ sudo cpio -icvt < /dev/st0
drwxr-xr-x  2 gqxing  gqxing          0   Nov  9 23:58 .
-rw-r--r--  1 gqxing  gqxing  104857600   Jun 30 11:30 data.0630
-rw-r--r--  1 gqxing  gqxing  104859648   Jul 31 11:30 data.0731
-rw-r--r--  1 gqxing  gqxing  104865792   Oct 31 11:30 data.1031
-rw-r--r--  1 gqxing  gqxing  104861696   Aug 31 11:30 data.0831
-rw-r--r--  1 gqxing  gqxing  104863744   Sep 30 11:30 data.0930
1024050 blocks
$
```

6. 使用cpio命令实现档案文件的追加备份

除了上述基本功能之外，利用“-A”选项，cpio命令还可提供档案文件的追加功能。在创建一个cpio档案文件之后，如果其中遗漏了部分文件，因而需要补充归档时，可以重新创建档案文件。如果档案文件较大，也可以利用cpio命令，把遗漏的文件附加到已建档案文件的后部，构成一个完整的档案文件。例如，假定之前已经备份了data目录中的数据文件，但忘记了备份主目录中的日志文件，可以使用下列命令，把主目录中的日志文件合并备份到原档案文件中：

```

$ cd /home/gqxing/data
$ find . -print | cpio -ocvB > /backup/data.cpio
.
./data.0630
./data.0731
./data.0831
./data.0930
./data.1031
102405 blocks
$ find ../backup.log -print | cpio -oAcv -F /backup/data.cpio
../backup.log
20 block
$ cpio -icvt < /backup/data.cpio
drwxr-xr-x 2 gqxing gqxing          0 Nov   9 23:58 .
-rw-r--r-- 1 gqxing gqxing 104857600 Jun  30 11:30 data.0630
-rw-r--r-- 1 gqxing gqxing 104859648 Jul  31 11:30 data.0731
-rw-r--r-- 1 gqxing gqxing 104865792 Oct  31 11:30 data.1031
-rw-r--r-- 1 gqxing gqxing 104861696 Aug  31 11:30 data.0831
-rw-r--r-- 1 gqxing gqxing 104863744 Sep  30 11:30 data.0930
-rw-r--r-- 1 gqxing gqxing   9699 Oct  01 08:26 ../backup.log
1024070 blocks
$

```

12.2.2 利用tar命令实现数据备份与恢复

tar命令可用于备份数据，把指定的文件、目录或文件系统及其中的所有文件组合后生成一个新的档案文件。生成的档案文件可以存储到磁带上，也可以存储到磁盘文件系统中。用户可以检索档案文件，必要时也可以把档案文件恢复到系统。在Linux系统中，tar还具有压缩功能，可以生成压缩的档案文件，因而能够采用ftp等网络通信工具，以二进制数据的形式实现系统间的文件复制与交换。

tar具有8种运行模式，或者说具有8种用途。

- 创建，利用“-c”选项，创建一个包含多个文件的档案文件，实现文件的备份。
- 替换，利用“-r”选项，把新文件写入档案文件后部，以便能够在恢复文件时实现新老文件的替换。
- 显示，利用“-t”选项，显示档案文件中的文件列表。
- 更新，利用“-u”选项，更新档案文件中的指定文件。
- 抽取，利用“-x”选项，把档案文件恢复到系统中。
- 合并，利用“-A”选项，把一个档案文件附加到另外一个档案文件的后面，实现档案文件的合并。
- 比较，利用“-d”选项，比较档案文件与磁盘中的文件。
- 删除，利用“-delete”选项，从档案文件中删除指定的文件。

tar命令的语法格式简写如下。表12-6给出了常用的tar命令选项及简单说明。

```

tar [-]c [-jMNvzZ] [-b n] [-f tarfile] [-F file] [-L n] [-X file] file-list
tar [-]r [-v] [-f tarfile] file-list
tar [-]t [-jMNvzZ] [-f tarfile] [-F file] [-X file] [file-list]

```

```

tar [-]u [-v] [-f tarfile] file-list
tar [-]x [-jmMNovzZ] [-f tarfile] [-F file] [-X file] [file-list]
tar [-]A tarfile1 -f tarfile2
tar [-]d -f tarfile
tar --delete delete-file -f tarfile

```

表12-6 tar命令的常用选项

选项	GNU选项	简单说明
-c	--create	创建。组合指定的文件，创建一个新的档案文件。如果同名的档案文件已经存在，在创建新的档案文件之前将会清除原有的档案文件
-r	--append	替换。把指定的文件写到档案文件的后部。采用这个选项时，原有的同名文件将会继续保留在档案文件中，并与新的文件共存于档案文件内。当使用“-x”选项抽取文件时，最新的同名文件将会取代原有的文件而保留下来
-t	--list	显示。显示档案文件中的文件列表。文件列表的输出格式类似于“ls -l”命令。如果使用“-t”选项时未指定文件参数，tar将会显示档案文件中的所有文件。如果指定了文件参数，tar只会列出与文件参数匹配的文件
-u	--update	更新。如果档案文件中不存在，或指定的文件比档案文件中的同名文件更新，则把指定的文件写到档案文件的后部。由于这个选项要求执行更多的检查，故其运行速度相对较慢
-x	--extract或--get	抽取或恢复文件。从档案文件中抽取指定的文件，按照档案文件的目录结构，复制到当前工作目录中。如果使用“-x”选项时未指定文件参数，tar将会抽取档案文件中的所有文件。如果指定了文件参数，tar只会抽出与文件参数匹配的文件。如果指定的文件参数含有目录，tar除抽取匹配的目录之外，还将递归地抽取目录中的所有文件。如果档案文件中包含若干同名的文件，后抽取的文件将会覆盖先前抽取的文件。在抽取文件时，tar将会尽可能地使抽取文件的文件属主、修改时间和访问权限等属性与档案文件中的初始文件保持一致。注意，在使用“-c”选项创建档案文件时，应尽可能采用相对路径名，否则，tar命令可能无法正确地匹配文件
-A	--catenate或--concatenate	把第一个档案文件附加到第二个档案文件后面，最终形成一个合并的档案文件
	--atime-preserve	在读写文件时不改变文件的访问时间
-d	--diff或--compare	比较档案文件与磁盘中的文件，检查两者之间是否存在差别，如文件大小和修改时间等
	--delete	从档案文件中删除指定的文件（不适用于磁带档案文件）
-b n	--blocking-size=n	在访问存储介质时，采用“n×512”个字节的逻辑数据块（n的默认值为20）作为读写单位，创建或读取档案文件。通常，在读取档案文件时，tar将会自动地确定逻辑数据块的大小
	--exclude=pattern	排除匹配指定模式的文件
-f tarfile	--file=tarfile	指定档案文件。档案文件可以是一个磁盘文件，也可以是一个设备文件（如磁带机等）
-F file	--info-script=file或 --new-volume-script=file	在读写到每个存储介质（如磁带）的结尾时运行指定的脚本文件（蕴含着使用“-M”选项）
	--help	给出tar命令及其有关选项的简单说明

(续表)

选项	GNU选项	简单说明
-j	--bzip2	创建和抽取档案文件时分别采用bzip2和bunzip2压缩或解压档案文件
-L <i>n</i>	--tape-length= <i>n</i>	指定存储介质的容量或档案文件的大小, 以KB为单位(即“ $n \times 1024$ ”个字节)。如果复制的档案文件大于这个选项指定的容量, 当写出 <i>n</i> -KB的数据时, tar将会提示用户更换新的存储介质(如磁带), 从而能够把档案文件分布存储到多个介质中。对于具有固定容量的存储介质(如磁带和软盘), 这个选项是非常有用的, 可用于创建多卷形式的档案文件
-m	--touch	采用抽取文件时的时间作为文件的修改时间。如果未指定“-m”选项, tar将恢复文件在档案文件中的初始修改时间。这个选项仅适用于“-x”选项
-M	--multi-volume	创建/显示/抽取多卷档案文件
-N	--after-date <i>date</i> 或 --newer <i>date</i>	仅读写比指定时间更新的文件。指定日期时可采用yyyymmdd或yyyymm-dd等形式表示年月日, 指定时间时可采用hhmm或hh:mm:ss的形式分别表示时分和时分秒
	--newer-mtime <i>date</i>	仅读写在指定时间之后其内容已经发生变动的文件。参见“-n”选项
-o	--no-same-owner	以执行tar命令的用户及其所属的用户组作为文件恢复后的用户和用户组, 而不采用档案文件中文件原有的用户和用户组属性。对于普通用户而言, 这是tar命令的默认处理方式。如果是超级用户运行tar命令, 且没有指定“-o”选项, 抽取的文件将采用档案文件中文件原有的用户和用户组属性。这个选项仅适用于“-x”选项
	--totals	创建档案文件之后显示已写出的字节总数
-v	--verbose	显示处理过程中读写的每一个文件名。与“-t”选项一起使用时, tar将会列出文件的详细属性信息, 如文件属主、访问权限和文件大小等, 其输出结果类似于“ls -l”命令的输出
	--wildcards	在指定文件模式时, 允许使用星号“*”等元字符
-X <i>file</i>	--exclude-from= <i>file</i>	表示归档或抽取指定文件中未列举的其他所有文件(即排除指定文件中包含的任何文件)。如果指定的文件是一个目录, 则排除指定的目录及其中的任何文件和子目录
	--exclude= <i>file</i>	表示归档或抽取除了指定文件之外的其他所有文件(即排除指定的任何文件)。如果指定的文件是一个目录, 则排除指定的目录及其中的任何文件和子目录
-z	--gzip或--gunzip	创建和抽取档案文件时分别采用gzip和gunzip压缩或解压档案文件
-Z	--compress或--uncompress	创建和抽取档案文件时分别采用compress和uncompress压缩或解压档案文件

假定想要备份/home/gqxing/src目录中的所有文件, 把归档后生成的档案文件存放在/share目录中, 可以使用下列tar命令:

```
$ cd /home/gqxing
$ tar -cvf /share/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
```

```
src/handler.c
src/Makefile
src/modules.c
src/listener.c
$
```

如果生成的atmsrc.tar档案文件过大，可以选用“-j”或“-z”等选项，使tar能够调用bzip2、gzip或gunzip等工具压缩档案文件，最终生成一个压缩的档案文件，例如：

```
$ ls -l /share/*.tar
-rw-r--r-- 1 gqxing gqxing 307200 Dec 17 13:37 /share/atmsrc.tar
$ tar -cjvf /share/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/modules.c
src/listener.c
$ ls -l /share/*.tar
-rw-r--r-- 1 gqxing gqxing 21217 Dec 17 13:41 /share/atmsrc.tar
$
```

必要时，可以从备份的档案文件中恢复指定的文件，或恢复全部文件。如果需要恢复归档的全部文件，可以使用下列tar命令：

```
$ cd /newdir
$ tar -xvf /share/atmsrc.tar
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/modules.c
src/listener.c
$
```

下面以1.44MB软盘（其设备名为/dev/fd0）和磁盘档案文件为例，说明怎样使用tar命令备份或恢复单个文件、一组文件，甚至整个文件系统。实际上，这里介绍的备份与恢复过程同样也适用于其他存储介质（如磁带、磁盘或CD/DVD等），只需把设备文件名改为相应存储介质的设备文件名即可。

1. 利用tar命令把文件归档备份到软盘上

在使用tar命令把文件复制到存储介质上时，应当注意下列事项：

- 使用tar命令的“-c”选项创建档案文件时，将会清除存储介质上原有的任何文件。
- 在复制文件时，可以使用文件名生成元字符“?”、“*”和“[...]”指定文件名。例如，为了复制所有以“.doc”为后缀的文件，可以使用“*.doc”作为文件名参数。
- 在抽取文件时，也可以使用文件名生成元字符“?”、“*”和“[...]”，从tar档案文件中抽取匹配的文件。
- 包含tar档案文件的存储介质不能作为文件系统安装。

为了使用tar命令把档案文件复制到软盘上，首先应进入需要复制文件的目录，然后使用下列tar命令复制文件：

```
tar -cvf /dev/fd0 filenames
```

其中，“-c”选项表示创建一个档案文件。“-v”选项表示在屏幕（标准输出）上显示正在处理的每一个文件及其相关的属性信息。“-f/dev/fd0”表示把创建的档案文件写到软盘中。filenames是准备归档的文件或目录名。示例如下：

```
$ cd /home/gqxing
$ ls src
Makefile atmcom.c atmmon.c handler.c listener.c modules.c
$ tar -cvf /dev/fd0 src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/modules.c
src/listener.c
$
```

为了验证指定的文件是否已经复制到软盘上，可以使用下列命令（其中，“-t”选项表示读取并显示档案文件中的文件列表）：

```
$ tar -tvf /dev/fd0
drwxr-xr-x  gqxing/gqxing      0 2009-12-17 13:35 src/
-rw-r--r--  gqxing/gqxing 30235 2009-11-17 17:56 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 18:03 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 55515 2009-11-17 18:00 src/modules.c
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
$
```

2. 利用tar命令把文件归档备份到多个软盘上

当使用tar命令把档案文件写入软盘时，应注意整个档案文件的大小和软盘的容量。如果数据量过大，一个软盘无法容纳所有的数据，因而需要使用多个软盘时，可以使用“-L”选项指定软盘的容量，以便能够在在一个软盘复制完成之后，由系统提示用户更换新的软盘。这里使用“-L 1440”选项指定软盘的容量为1.44MB。

```
$ cd /opt/atm
$ tar -L 1440 -cvf /dev/fd0 doc
doc/
doc/user-guide
doc/reference-manual
Prepare volume #2 for '/dev/fd0' and hit return:
doc/release-notes
doc/technical-specifications
$
```

当复制到中途时，系统会提醒用户更换新的软盘。在更换下一个软盘之后，按下“Enter”

键，系统将会继续执行文件的复制，直至把所有的文件均写入软盘中。

3. 利用tar命令恢复档案文件中的所有文件

为了从先前备份到软盘的档案文件中恢复所有的文件，首先应进入某个选定的目录，然后使用下列命令抽取其中的文件：

```
tar -xvf /dev/fd0 [filenames]
```

其中，“-x”选项表示从指定的档案文件中抽取文件，并把抽取的文件恢复到当前工作目录中。“-v”选项表示在屏幕（标准输出）上显示抽取的每一个文件及其相关的属性信息。

“f/dev/fd0”表示软盘设备。Filenames是准备抽取的文件。如果同时指定多个文件，文件名之间应加空格字符。默认情况下（未指定文件名参数），tar命令将会抽取所有的文件。示例如下：

```
$ su -c cathy
Password:
$ tar -xvf /dev/fd0
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/modules.c
src/listener.c
$ ls src
Makefile atmcom.c atmmon.c handler.c listener.c modules.c
$
```

4. 利用tar命令抽取档案文件中的指定文件

如前所述，在“tar -x”命令中，如果命令行中未指定文件参数，则意味着抽取档案文件中的所有文件。为了抽取特定的文件，需要在命令行中指定文件名。指定文件名时必须严格地匹配档案文件中的文件名。如果无法肯定文件名或路径名，可以使用下面下一小节将要介绍的过程，获取档案文件中的完整文件列表。

假定需要从先前备份的档案文件中抽取atmmon.c文件，可以使用下列命令：

```
$ cd /newdir
$ tar -xvf /dev/fd0 src/atmmon.c
src/atmmon.c
$ ls -l src
total 60
-rw-r--r-- 1 qqxing qqxing 60978 Nov 17 18:03 atmmon.c
$
```

在tar命令中，也可以使用文件名生成元字符指定欲抽取的文件。例如，为了抽取档案文件中所有以“.c”为后缀的文件，可以使用下列命令：

```
$ tar --wildcards -xvf /dev/fd0 *.c
src/atmcom.c
src/atmmon.c
src/handler.c
src/modules.c
src/listener.c
$
```

但在使用“?”、“*”和“[...]”文件名生成元字符时，需要使用准确的匹配模式，才能够抽取期望的文件。例如，假定新建的档案文件中还存在一个Makefile或makefile文件，在抽取这个文件时，可以使用“src/[Mm]*”作为匹配模式，但不能简单地使用的“[Mm]*”，这是因为tar把整个目录和文件的路径名看做一体。示例如下：

```
$ tar -xvf /dev/fd0 src/[Mm]*
src/Makefile
$
```

5. 利用tar命令查询先前备份的档案文件

为了查询先前备份的档案文件，可以使用下列命令：

```
tar -tvf /dev/fd0
```

其中，“-t”选项表示读取并显示档案文件中的文件列表。“-v”选项表示在屏幕（标准输出）上显示每一个文件及其相关的属性信息。“-f/dev/fd0”表示软盘设备。例如：

```
$ tar -tvf /dev/fd0
drwxr-xr-x  gqxing/gqxing      0 2009-12-17 13:35 src/
-rw-r--r--  gqxing/gqxing 30235 2009-11-17 17:56 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 18:03 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 55515 2009-11-17 18:00 src/modules.c
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
$
```

6. 利用tar命令更新先前备份的档案文件

利用“-u”选项，tar命令支持档案文件的更新。在创建一个tar档案文件之后，如果其中的部分文件后来发生了变动，因而也需要归档备份时，可以重新创建档案文件。如果档案文件较大，也可以利用tar命令仅仅更新发生变动的文件，把变动的文件附加到原档案文件的后部（原有的老文件仍然保留）。例如，假定之前已经备份了src目录中的所有源文件，之后又改动了其中的atmcom.c文件，可以使用下列命令更新档案文件：

```
$ tar -cvf /tmp/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/module.c
src/listener.c
$ touch src/atmcom.c
$ tar -uvf /tmp/atmsrc.tar src/atmcom.c
src/atmcom.c
$ tar -tvf /tmp/atmsrc.tar
drwxr-xr-x  gqxing/gqxing      0 2009-12-17 13:57 src/
-rw-r--r--  gqxing/gqxing 30235 2009-11-17 11:17 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 11:19 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
```

```
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 55515 2009-11-17 11:10 src/module.c
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
-rw-r--r--  gqxing/gqxing 30235 2010-01-23 14:08 src/atmcom.c
$
```



注意 tar命令的更新功能不支持压缩的档案文件，因此，如果先前采用“-j”、“-z”或“-Z”选项创建了压缩的档案文件，只能是重建新的档案文件。

7. 利用tar命令实现档案文件的追加备份

利用“-r”选项，tar命令提供档案文件的追加功能。在创建一个tar档案文件之后，如果其中遗漏了部分文件，因而需要补充归档时，可以重新创建档案文件。如果档案文件较大，也可以利用tar命令，把遗漏的文件附加到原档案文件的后部，构成一个完整的档案文件。例如，假定之前已经备份了src目录中的源文件，但忘记了备份incl目录中的头文件，可以使用下列命令，把incl目录中的文件合并备份到原档案文件中：

```
$ tar -cvf /tmp/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/module.c
src/listener.c
$ tar -rvf /tmp/atmsrc.tar incl
incl/
incl/atmdef.h
$ tar -tvf /tmp/atmsrc.tar
drwxr-xr-x  gqxing/gqxing      0 2009-12-17 13:57 src/
-rw-r--r--  gqxing/gqxing 30235 2010-01-23 14:08 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 11:19 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 55515 2009-11-17 11:10 src/module.c
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
drwxr-xr-x  gqxing/gqxing      0 2009-12-17 14:26 incl/
-rw-r--r--  gqxing/gqxing 10498 2009-11-17 14:26 incl/atmdef.h
$
```



注意 tar命令的补充备份功能不支持压缩的档案文件。

8. 利用tar命令合并两个档案文件

利用tar命令的“-A”选项，可以把一个档案文件附加到另一个档案文件的后面。例如，假定之前已经备份了src目录中的源文件，又单独备份了incl目录中的头文件，使用下列命令，可以把第二个档案文件合并到第一个档案文件中：

```
$ tar -cvf /tmp/atmsrc.tar src
src/
```

```

src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/module.c
src/listener.c
$ tar -cvf /tmp/atmhdr.tar incl
incl/
incl/atmdef.h
$ tar -A /tmp/atmhdr.tar -f /tmp/atmsrc.tar
$ tar -tvf /tmp/atmsrc.tar
drwxr-xr-x  gqxing/gqxing    0 2009-12-17 13:57 src/
-rw-r--r--  gqxing/gqxing 30235 2010-01-23 14:08 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 11:19 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 55515 2009-11-17 11:10 src/module.c
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
drwxr-xr-x  gqxing/gqxing    0 2009-12-17 14:26 incl/
-rw-r--r--  gqxing/gqxing 10498 2009-11-17 14:26 incl/atmdef.h
$

```



注意 tar命令的档案文件合并功能不支持压缩的档案文件。

9. 利用tar命令删除档案文件中的指定文件

利用tar命令的“—delete”选项，可以从档案文件中删除指定的文件。例如，假定之前已经备份了src目录中的源文件，为了删除其中的module.c文件，可以使用下列命令：

```

$ tar -cvf /tmp/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/module.c
src/listener.c
$ tar --delete src/module.c -f /tmp/atmsrc.tar
$ tar -tvf /tmp/atmsrc.tar
drwxr-xr-x  gqxing/gqxing    0 2009-12-17 13:57 src/
-rw-r--r--  gqxing/gqxing 30235 2010-01-23 14:08 src/atmcom.c
-rw-r--r--  gqxing/gqxing 60978 2009-11-17 11:19 src/atmmon.c
-rw-r--r--  gqxing/gqxing 82468 2009-11-17 18:04 src/handler.c
-rw-r--r--  gqxing/gqxing 10088 2009-11-17 18:00 src/Makefile
-rw-r--r--  gqxing/gqxing 58257 2009-11-17 18:02 src/listener.c
$

```



注意 tar命令的文件删除功能不支持压缩的档案文件。

10. 利用tar命令比较档案文件

利用tar命令的“-d”选项，可以根据磁盘中的文件，逐一比较档案文件中的文件，检查备份的档案文件是否过期。例如，假定之前已经备份了src目录中的源文件，利用下列命令可以比较两者之间的差异（最后一个tar命令说明Makefile文件的修改时间有异）：

```
$ tar -cvf /tmp/atmsrc.tar src
src/
src/atmcom.c
src/atmmon.c
src/handler.c
src/Makefile
src/module.c
src/listener.c
$ tar -d -f /tmp/atmsrc.tar
$ touch src/Makefile
$ tar -d -f /tmp/atmsrc.tar
src/Makefile: Mod time differs
$
```



注意 tar命令的文件比较功能不支持压缩的档案文件。

12.2.3 利用dd命令实现数据的原样复制

dd命令的最大特点是能够采用原始读写的方式，逐块逐道地把位于存储介质上的数据原封不动地复制到另一个存储介质上。此外还可以把一个文件系统完整地复制到另一个磁盘分区中，从而实现文件系统的备份。通常，dd命令的功能是把标准输入复制到标准输出中。

在复制过程中，dd命令还具有数据转换功能，可以把不同代码格式的数据转换为另一种编码的数据，实现不同系统之间的数据交换。

与其他大多数命令相比，dd命令的语法明显不同。在dd命令中，命令选项采用“keyword=value”的形式，其语法格式简写如下：

```
dd keyword=value .....
```

其中，keyword相当于其他命令中的选项，value是选项的参数。当需要复制某个文件时，可以采用下列简单的语法格式：

```
dd if=input-file of=output-file
```

表12-7 给出了dd命令的部分常用选项和参数及说明。

表12-7 dd命令的部分常用选项和参数

选项与参数	简单说明
if=file	指定输入文件。文件可以是普通数据文件，也可以是磁盘分区、磁带、软盘或CD/DVD等设备文件。默认值为标准输入
of=file	指定输出文件。文件可以是普通数据文件，也可以是磁盘分区、磁带、软盘或CD/DVD等设备文件。默认值为标准输出

(续表)

选项与参数	简单说明
<code>ibs=<i>n</i></code>	指定输入数据块的大小，数值后面可以附加一个表示数据块单位的字符，如b、K、M、G和T等，分别表示以 <i>n</i> 个512字节、1KB、1MB、1GB和1TB的数据块为数据输入单位。如果未指定表示数据块单位的字符，则以字节为计数单位。如果均未指定，默认值为512个字节
<code>obs=<i>n</i></code>	指定输出数据块的大小，数值后面可以附加一个表示数据块单位的字符，如b、K、M、G和T等，分别表示以 <i>n</i> 个512字节、1KB、1MB、1GB和1TB的数据块为数据输出单位。如果未指定表示数据块单位的字符，则以字节为计数单位。如果均未指定，默认值为512个字节
<code>bs=<i>n</i></code>	指定读写数据块的大小，数值后面可以附加一个表示数据块单位的字符，如b、K、M、G和T等，分别表示以 <i>n</i> 个512字节、1KB、1MB、1GB和1TB的数据块为输入单位。如果未指定表示数据块单位的字符，则以字节为计数单位。如果均未指定，默认值为512个字节。此选项可以同时强制替代ibs和obs两个选项
<code>skip=<i>n</i></code>	在开始复制之前，先从输入文件的起始位置跳过指定数量的数据块（以ibs或bs定义的输入数据块为计数单位）
<code>seek=<i>n</i></code>	在开始复制之前，先从输出文件的起始位置跳过指定数量的数据块（以obs或bs定义的输出数据块为计数单位）
<code>count=<i>n</i></code>	指定复制的数据块数量。每个数据块的字节数以ibs、obs或bs的定义为准

例如，若想复制一个软盘，可以使用dd命令先把软盘复制到一个临时文件中：

```
$ dd if=/dev/fd0 of=/tmp/tmpfile
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied. 48.6294 seconds. 30.3 kB/s
$
```

然后，再使用dd命令，交换输入和输出文件的位置，把临时文件复制到一个新的软盘中：

```
$ dd if=/tmp/tmpfile of=/dev/fd0
2880+0 records in
2880+0 records out
1474560 bytes (1.5 MB) copied. 98.822 seconds. 14.9 kB/s
$
```

运行结束之后，dd命令将会给出已经读写的数据块数量。其中，加号“+”之前的数字表示复制过程中读写的完整数据块数量。加号“+”之后的数字表示复制了多少个不完整的数据块。

若想把磁盘分区2中的文件系统复制到磁盘分区3中，可以使用下列命令：

```
$ sudo dd if=/dev/sda2 of=/dev/sda3 bs=1024K
xxxx+0 records in
xxxx+0 records out
xxxxxxxxxxx bytes (x.x GB) copied. xxx.xxx seconds. x.x MB/s
$
```

然后，可以使用“fsck -f”命令，检测新复制的文件系统。最后再使用mount命令安装新的文件系统。

```
$ sudo fsck -f /dev/sda3
.....
$ sudo mount /dev/sda3 /mnt
$
```

在使用dd命令复制磁盘分区（文件系统）时，应注意下列事项：

- 确保源磁盘分区与目的磁盘分区具有相同的容量（且最好为同一型号的磁盘）。
- 使用fsck命令检测新复制的文件系统。然后利用mount命令予以检验。
- 使用dd命令复制磁盘分区时，应确保系统处于单用户运行模式，从而保证源磁盘分区中的数据不会发生变化。

同样，dd命令也可用于复制磁带。如果系统配有两个磁带机，则可以使用下列命令，实现从磁带到磁带的直接复制。否则，需要先把数据复制到磁盘上，然后再复制到新的磁带中。

```
$ dd if=/dev/st0 of=/dev/st1
```

12.3 采用专用工具备份与恢复数据

除了cpio、tar与dd等基本软件工具之外，Ubuntu Linux系统还提供诸如dump/restore、amanda、rsync、BackupPC以及bacula等专业备份工具，可以选择使用，但这些软件工具需要单独安装，简述如下：

- dump/restore: dump与restore是一对分工明确的数据备份与恢复工具。其中，dump用于备份数据，restore负责恢复数据。注意，dump与restore仅适用于Ext2/3/4文件系统，能够准确地备份与恢复其中的任何文件，如设备特殊文件等。此外，dump/restore只能用于备份与恢复整个文件系统，其备份方式包括完整备份与增量备份。dump/restore采用/etc/dumpdates文本文件，提供哪个文件系统已经备份，以及应执行的备份级别等信息。dump创建的档案文件能够跨越多个磁带等存储介质，restore能够以交互方式运行，且能够仿真备份数据的实际目录层次，供用户查询、选择准备恢复的目录与文件。
- amanda: amanda是“Advanced Maryland Automated Network Disk Archiver”的英文缩写，由马里兰大学开发。利用amanda，能够非常容易地把任何远程系统的数据集中备份到中心服务器，同时提供完整的管理功能。采用不同的配置，amanda能够支持多种备份方式与手段，生成详细的备份报告与日志信息，能够通过电子邮件，自动提交备份报告。中心服务器与远程系统的数据通信采用加密方式，从而确保数据的安全。amanda分为两个软件包：amanda-server和amanda-client，两个软件包均需单独安装。amanda的网站地址是<http://www.amanda.org>。
- rsync: rsync是一个命令行的文件与目录同步软件工具，便于用户在系统之间复制文件和目录。当本地与远程系统均存在文件和目录的副本时，rsync能够有效地减少传输的数据量，确保本地与远程系统的文件和目录副本是等同的。rsync采用远程更新协议，能够确保仅仅传输两组文件和目录副本的不同部分。rsync是Ubuntu Linux系统的基本组成部分之一，不需要单独安装，但在运行前需要对准备在本地系统备份的远程系统进行适当的配置。
- BackupPC: BackupPC提供一个基于浏览器界面的备份工具，能够利用Samba、tar或rsync

备份远程系统。BackupPC创建的远程系统备份可在BackupPC服务器中集中存储和管理。授权的用户能够从中恢复自己的文件。针对每一个系统，BackupPC服务器能够分别存储不同的配置数据，从而能够针对每一个系统，采用不同的命令、协议与备份方式，实现不同类型的数据备份。BackupPC的另外一个特点是采用标准的Linux系统命令，创建数据备份，因而远程系统不需要安装额外的软件。有关BackupPC软件工具的详情，可以访问<http://backuppc.sourceforge.net>网站。

- bacula: bacula包含一组强有力的软件与文档，提供网络备份功能，支持Linux、UNIX和Windows系统。与amanda相比，在如何存储及存储位置的选择方面，bacula显得更灵活。正由于其功能强大，使用起来也比较复杂。尽管提供一个图形界面的控制台，但bacula仍然是一个命令行软件工具。有关bacula软件工具的使用说明，详见<http://www.bacula.org>网站。

在上述备份软件工具中，dump/restore只能用于本地系统的数据备份，其他软件工具可用于网络环境的数据备份。

这一节以dump/restore命令为例，介绍系统数据的备份与恢复过程。除了少数的增量备份之外，在实施大数据量的备份时，磁带、磁盘或CD/DVD是必不可少的备份介质。

在开始备份数据之前，通常应事先采用df、du或dump等命令，对需要备份的文件系统或磁盘分区的数据量进行评估，以确定应采用何种类型的备份介质，以及需要准备多少存储介质。

在安装且适当设置了全部系统软件之后，可以从根目录开始，对整个系统做一个完整的备份，或对业务数据所在文件系统单独做一个完整的备份。之后，只需定期备份/var、/home和业务数据所在的文件系统。一旦系统受损，可以利用系统或业务数据的完整备份或增量备份，恢复系统与业务数据。

在系统运行期间，/var、/home和业务数据所在的文件系统经常会频频发生变动。因此，最好每周执行一次完整的文件系统备份，每天执行一次增量备份。

12.3.1 利用dump命令备份数据

dump命令用于备份指定的文件系统，把整个文件系统中的目录与文件写入指定的备份介质中，如磁带、磁盘或CD/DVD等。也可以采用增量备份方式，仅仅备份一定时间内发生变动的文件。dump命令的语法格式简写如下：

```
dump [-level#] [-aMSu] [-A file] [-B records] [-b blocksize] [-D file]
[-f file] [-F script] [-L label] [-T date] files-to-dump
```

dump的命令选项主要用于指定备份方式（如完整备份或增量备份）、备份目的（如备份文件或备份介质）、备份文件列表、备份记录的数据块数量、脚本文件、介质卷标、多卷备份以及其他备份要求。表12-8给出了部分常用的选项及简单说明。files-to-dump可以是一个文件系统的安装点，也可以是作为文件系统子集的一组文件与目录列表，用于指定想要备份的数据。

表12-8 dump命令的部分常用选项

选项	简单说明
-level#	备份级别。备份级别可以是任何整数（之前，合法的备份级别为0~9，现可为任意整数），其中0表示完整备份整个文件系统。大于0的任何备份级别均为增量备份，这意味着仅仅备份自上一次执行较低级别的备份以来新建或修改过的所有文件。默认的备份级别为9

选项	简单说明
-a	表示绕过计算存储介质容量，只管写入存储介质，直至遇到介质结束标志
-A <i>file</i>	生成一个备份文件列表，以便restore命令能够确定准备恢复的文件是否位于备份介质中
-b <i>blocksize</i>	指定每个备份记录包含多少1 KB的数据块。通常，每个备份记录包含10个1 KB的数据块
-B <i>records</i>	指定每个存储介质能够容纳的1 KB数据块的数量。通常不需要指定，dump能够检测存储介质的结束标志。当达到指定的容量或写满存储介质时，dump将会提示用户更换介质
-D <i>file</i>	指定用于存储完整备份或增量备份信息的文件。默认的文件为/etc/dumpdates
-f <i>file</i>	把备份数据写到指定的文件。文件参数可以是设备文件（如/dev/st0磁带机）、普通文件或减号“-”（标准输出）。若想指定多个文件，文件名之间需加逗号“,”分隔符。每个文件可以按给定的顺序作为多卷备份中的一个备份卷。如果指定的文件为“host:file”或“user@host:file”，dump将会利用rmt命令，把备份数据写入远程主机中的指定文件（文件事先必须存在，dump不会创建新的文件）。远程rmt命令默认的路径名为/etc/rmt（必要时可以使用环境变量RMT重新定义）
-F <i>script</i>	在写入每个备份卷（最后一个除外）结束之后，运行指定的脚本。dump将会把设备文件名和备份卷的卷号作为命令行参数传递给指定的脚本。脚本运行结束之后，如果返回值为0，意味着dump可以继续执行备份，无需请求用户更换磁带。如果脚本的返回值为1，dump将会提示用户更换介质，然后才能继续执行备份。其他返回值将会导致dump终止执行
-L <i>label</i>	指定的字符串可以写入备份介质的头部，作为备份介质的卷标，使restore和file等工具软件能够快速访问及识别文件的类型。当前，字符串的长度限于16个字符（包括终止符“\0”）
-M	支持多卷备份功能。“-f”选项指定的文件名将会作为一个前缀，用于命名一系列备份文件，如<filename>001、<filename>002等。dump将会依次写入每个备份文件。在Ext2文件系统中，为了存储备份文件，同时克服2 GB文件大小的限制，这个选项是非常有用的
-S	显示指定文件系统、目录或文件的字节数量，以便估算存储空间需求。在开始备份之前，可用这个选项确定存储空间的容量需求，从而确定需要多少备份介质
-T <i>date</i>	使用指定的时间（而不是/etc/dumpdates文件中的时间）作为起始备份时间，时间格式为“www mmm dd hh:mm:ss yyyy”，如“Thu Nov 12 00:55:05 2009”，时区偏移值格式为“± HHMM”，如+0800（北京时间）。如果未指定时区偏移值，可以把备份时间看做本地时间。注意，“-T”选项不能与“-u”选项一起使用
-u	在成功地完成备份之后，更新/etc/dumpdates文件。文件的记录格式如下： <文件系统名> <增量备份级别> <备份时间> <时区偏移值>

如前所述，事先了解备份的数据量是非常有用的，这有助于确定采用什么备份介质，以及需要准备多少存储介质。例如，为了确定/appdata文件系统的数据量，以便制作一个业务数据的完整备份，可以利用dump命令的“-S”选项：

```
$ sudo dump -S /appdata
4026958848
$
```

上述dump命令的返回结果4026958848表示指定文件系统现有数据的字节数，经计算约为4 GB，这意味着至少需要使用一个7 GB的DLT磁带。为了制作一个备份级别为0的完整备份，把数据写到一个7 GB的磁带上，且在完成备份后更新备份记录文件/etc/dumpdates，可以使用下列命令：

```
$ sudo dump -0u -f /dev/st0 /appdata
DUMP: Date of this level 0 dump: Mon Nov 16 22:00:09 2009
DUMP: Dumping /dev/sdb6 (/appdata) to /dev/st0
DUMP: Label: /appdata
DUMP: Writing 10 Kilobyte records
.....
$
```

在制作了完整备份之后，可以选用一个大于0的备份级别，采用下列dump命令，对发生变动的文件或数据，定期执行增量备份：

```
$ sudo dump -9u -f /dev/st0 /appdata
DUMP: Date of this level 9 dump: Mon Nov 16 22:05:10 2009
DUMP: Date of last level 0 dump: Mon Nov 16 22:00:09 2009
DUMP: Dumping /dev/sdb6 (/appdata) to /dev/st0
DUMP: Label: /appdata
DUMP: Writing 10 Kilobyte records
.....
$
```

利用“-f”选项，dump命令还支持远程备份。此外，当备份数据大于单个存储介质的容量时，可以采用多卷备份方式，把数据分布于多个存储介质上。



在运行dump命令时，文件系统应处于静止状态，否则无法确保当前文件系统与实际备份数据的一致性，从而导致restore命令无法正确地恢复文件。一个文件系统能够保持静止状态，仅当文件系统已经卸载，或系统处于单用户运行模式。

12.3.2 利用restore命令恢复数据

在恢复备份的数据时，需要准确地知道究竟要恢复多少数据，恢复哪些数据。如果选择完整恢复，可能会浪费时间。如果恢复选定的文件，必须确保已经恢复了受到影响的所有文件。在一个业务系统中，通常也许只需恢复最近一次的增量备份，而在其他情况下，可能还需要恢复最近一次的完整备份，然后再依次恢复之后的增量备份。在恢复期间，切忌恢复不必要的，且其数据已经过时的文件。

restore命令用于恢复先前采用dump命令制作的任何备份数据，如文件系统的完整备份或增量备份。restore能够恢复单个文件、部分选定的文件或整个文件系统。恢复时可以先恢复完整的文件系统，在此基础上，再恢复之后制作的增量备份。此外，利用dump/restore的远程备份与恢复功能，还可以从远程主机中恢复备份的数据（参见“-f”选项）。restore命令的语法格式简写如下：

```
restore -C [options] [-b blocksize] [-D filesystem] [-f file]
restore -i [options] [-A file] [-b blocksize] [-f file]
restore -R [options] [-b blocksize] [-f file]
restore -r [options] [-b blocksize] [-f file]
restore -t [options] [-A file] [-b blocksize] [-f file] [-X filelist] [file]
restore -x [options] [-A file] [-b blocksize] [-f file] [-X filelist] [file]
```

在运行restore命令时，restore必须处于下列6种常用的运行模式之一，其他命令选项和参数是选用的。

- 比较文件，利用“-C”选项，能够比较备份介质与磁盘中的文件。
- 交互恢复，利用“-i”选项，能够采用交互方式浏览备份介质，恢复选定的文件。
- 断点恢复，利用“-R”选项，能够从指定的备份介质开始，执行先前中断的恢复。
- 完整恢复，利用“-r”选项，能够重建并恢复一个完整的文件系统。
- 显示文件，利用“-t”选项，可以从备份介质读取并显示文件列表。
- 部分恢复，利用“-x”选项，能够从指定的备份介质中恢复指定的文件。

在restore命令的语法格式中，“-C”、“-i”、“-R”、“-r”、“-t”或“-x”等选项用于确定restore命令的运行模式，运行时必选其一。文件参数file（包括“-f”选项中的文件参数file）用于指定准备恢复的文件、目录或文件系统的根目录。表12-9给出了restore命令的部分常用选项及简单说明。

表12-9 restore命令的部分常用选项

选项	简单说明
-C	利用这个选项，restore命令能够读取备份介质，并与系统中的现有文件进行比较。在开始比较之前，restore首先会把当前工作目录改换到备份文件系统的根目录，然后逐一比较系统与备份介质中的文件
-i	<p>采用交互方式，从备份介质中恢复文件。从备份介质中读取文件目录信息之后，restore将会提供一个命令行界面，使用户能够遍历目录树，从中选择想要恢复的文件。下面是部分可用的交互命令（对于需要提供参数的交互命令，默认的参数是当前目录）：</p> <ul style="list-style-type: none"> · add [arg]——把当前或指定目录及其中的所有文件（包括子目录中的文件）加到欲恢复的文件列表中，除非命令行中指定了“-h”选项。当利用ls交互命令显示文件时，位于欲恢复文件列表中的文件名之前冠有一个星号“*”前缀。 · cd arg——把当前目录改换到指定的目录。 · delete [arg]——从欲恢复的文件列表中删除当前或指定目录及其中的所有文件（包括子目录中的文件），除非命令行中指定了“-h”选项 · extract——根据欲恢复的文件列表，从备份介质中恢复指定的所有文件。执行前，restore将会询问从哪一个备份介质开始恢复文件。恢复部分文件时，最好能够从一个目标确定的备份介质开始（而非从头开始）恢复文件 · help——显示可用的交互命令及简单说明 · ls [arg]——列举当前或指定目录中的文件和目录。如果是一个目录，其名字后面将会附加一个“/”后缀。如果是一个欲恢复的文件或目录，其名字前面有一个星号“*”标记 · pwd——显示当前工作目录的完整路径名 · quit——退出restore命令
-R	从多卷备份的指定介质开始恢复（参见“-r”选项）。在全面恢复期间，如果restore的恢复过程因故中断，使用这个选项可以从断点处开始继续恢复
-r	重建或恢复一个文件系统。全面恢复时，restore将会利用mke2fs命令创建并安装一个新的文件系统，然后进入文件系统的根目录，递归地全面恢复采用备份级别0备份的整个文件系统。如果已经成功地恢复了一个完整备份，还可以利用“-r”选项，恢复任何必要的增量备份
-t	如果存在，列出备份介质中的指定文件。如果未指定文件参数，则从根目录开始，列出备份介质中的所有目录和文件，除非指定了“-h”选项。参见“-X”选项

(续表)

选项	简单说明
-x	从给定的备份介质中抽取指定的文件，并把文件恢复到系统中。如果指定的文件参数匹配备份介质中的某个目录，且未指定“-h”选项，则递归地抽取目录中的文件，并尽可能地恢复文件的属主、修改时间和访问权限等属性。如果未指定文件参数，则从根目录开始，恢复备份介质中的所有文件，除非指定了“-h”选项。参见“-X”选项
-a	当使用“-i”或“-x”选项恢复文件时，restore将会请求用户提供一个卷号，表示从哪一个备份介质开始恢复文件，这一做法能够直接读取目标备份介质，减少文件恢复的时间。“-a”选项表示禁止采用这种处理方式，即总是从第一个备份卷开始，依次恢复所有的备份介质。如果不知道究竟应从哪一个备份介质开始恢复文件，可以使用这个选项
-A <i>file</i>	从指定的文件而非备份介质中读取文件列表，以便在不访问备份介质的情况下，也能够确定欲恢复的文件是否存在于备份介质。这个选项可与“-t”、“-i”或“-x”选项一起组合使用
-b <i>blocksize</i>	指定每个备份记录包含多少个1KB的数据块。如果未指定“-b”选项，restore将会尝试自动确定存储介质中备份记录的数据块数量
-D <i>filesystem</i>	当采用“-C”选项比较备份介质时，可用“-D”选项指定文件系统的名字
-f <i>file</i>	从指定的文件中恢复备份的数据。文件参数可以是一个设备文件（如磁带机/dev/st0、磁盘分区/dev/sda2等）、普通文件或减号“-”（标准输出）。如果文件参数是一个形如“host:file”或“user@host:file”的远程文件，restore将会利用rmt命令，从远程主机中恢复指定的文件
-h	仅表示目录本身而不涉及其中的文件。这将防止restore递归地恢复整个目录层次子树
-M	多卷恢复。可用于恢复采用“-M”选项备份的多卷存储介质。“-f”选项指定的文件名将会作为一个前缀，使restore能够按照<filename>001、<filename>002、……的顺序，尝试读取多个存储介质或备份文件
-N	利用“-N”选项，restore能够执行“-i”、“-R”、“-r”、“-t”或“-x”选项定义的恢复动作，但实际上并不把任何文件写入磁盘中
-o	与“-i”或“-x”恢复模式不同，利用“-o”选项，restore能够自动恢复当前目录，而无需操作员干预
-u	恢复文件时，如果存在同名的文件，restore将会发出一个警告信息。为了避免这种情况出现，可以使用“-u”选项，使restore能够强行覆盖原有的文件
-v	在restore的恢复过程中，通常不会显示任何处理信息。“-v”选项能够使restore显示其处理的每一个文件，并在文件名之前冠以文件的类型
-V	支持非磁带（如CD/DVD）的多卷恢复功能
-X <i>filelist</i>	除了命令行中指定的文件之外，再从指定的文件中读取文件列表，组成一个完整的文件列表。这个选项可与“-t”或“-x”选项一起使用。在制作文件列表时，每个文件名占一行

如前所述，当使用“-i”选项调用restore命令时，可以进入交互恢复方式，此时可以浏览备份介质，查询其中的文件内容，确定欲恢复或放弃哪些文件。例如，下列命令将会启动一个交互恢复会话：

```
$ sudo restore -i -f /dev/st0
restore >
```

在“restore >”提示符下，可以使用cd和ls等交互命令，查询指定备份磁带（其设备文件名为/dev/st0）中的备份数据，从显示的文件列表中选择欲恢复的文件和目录。然后利用add交互命令，把选定的目录或文件加到欲恢复的文件列表中；或者利用delete命令，从文件列表中删除不准备恢复的目录或文件。最后使用extract交互命令，恢复选定的文件。在交互恢复方式中，也可以使用help交互命令查阅可用的命令，了解命令的简单用法。

除了“-C”、“-i”、“-R”、“-t”、“-T”和“-x”等运行模式选项之外，restore命令的其他选项可用于指定恢复的处理模式、恢复文件列表、读写数据块的大小、备份数据的位置、多卷恢复，以及显示命令的处理结果等。

例如，为了从指定的备份介质/dev/st0中，从头开始完整地恢复/appdata文件系统，不管是否存在同名的文件，强制执行文件恢复，可以使用下列命令：

```
$ sudo restore -rau -f /dev/st0
```

第13章 软件管理

本章主要介绍软件管理，以及Linux系统的更新与升级。讨论怎样利用Linux系统提供的软件包管理工具，安装、更新、升级以及查询系统软件。

13.1 软件管理概述

13.1.1 软件维护工具

Ubuntu Linux系统提供一系列命令行软件维护工具，如apt-get、aptitude以及dpkg等。这些软件维护工具各有侧重，但在安装软件包、更新以及升级Linux系统，确保系统总是处于最新状态方面，基本上具有等效的功能，而且都能够按照用户的意图，以网上发布的最新软件为准，安装或更新软件包及底层依赖的相关软件。为了确保系统的安全性，增强系统的可靠性与可用性，定期地更新系统是一项必不可少的系统维护任务。

Ubuntu Linux系统也提供若干图形界面的软件维护工具，用以实现软件的安装、更新以及系统升级。在GNOME桌面环境中，可以使用synaptic与gnome-app-install。synaptic是一个通用图形界面的软件维护工具，既可用于安装软件包，也可用于更新已安装的软件包，或者升级整个系统。gnome-app-install主要用于补充安装尚未安装的GNOME桌面软件包。

此外，Ubuntu Linux系统还提供一对软件更新工具update-notifier和update-manager，用于检测软件仓库中是否存在可用的软件包，现有系统中哪些软件包需要更新。当存在更新版本的软件包时，提醒用户更新系统中的相关软件，详见本章第13.6节。

13.1.2 软件管理基本概念

1. 软件包

在Ubuntu Linux系统中，所有软件与文档都是以软件包档案文件形式提供的。软件包可以分为二进制软件包（用于封装可执行程序、相关文档及配置文件等）和源代码软件包（其中主要包含源代码以及生成二进制软件包的方法）。

二进制软件包通常是一种压缩格式的档案文件，其中包含产品信息、程序文件、配置文件、随机文档、启动脚本及控制信息等，使用户能够容易地安装、更新、升级以及删除软件（注意，若想查询一个软件包中究竟包含哪些文件，可以使用“dpkg -L pkgname”命令）。

软件包管理工具可以利用其中的说明与控制信息，检索、安装、删除、更新或升级软件与系统。此外，除了基本的系统引导过程之外，Ubuntu安装程序也是利用Ubuntu项目组提供的软件包，按照用户的要求，安装Linux系统的。

按照封装格式，常见的Linux系统软件包可以分为下列三种类型（Ubuntu Linux系统主要支持Debian格式的软件包）：

- Debian格式软件包（文件扩展名为.deb）——Debian软件包是Linux系统中最常用的两种软件包之一，也是Debian及其派生Linux系统（包括Ubuntu）主要支持的标准软件包。

Ubuntu软件仓库中提供的软件包均采用这种封装格式，apt-get、aptitude以及synaptic等软件管理工具也均支持此类软件包。

- Red Hat格式软件包（文件扩展名为rpm）——RPM（Red hat Package Manager）是另外一种流行的Linux系统软件包，也是Red Hat及其派生Linux系统（如Fedora）支持的软件包。
- Tarball——是一种由大量文件（包括其目录结构）组装成单个档案文件的大型文件集合。其中，tar命令用于组合多个文件，生成一个档案文件，以便于发行；gzip命令用于压缩文件的容量，以节省文件的存储空间。Tarball非常类似于Windows系统中的.zip文件。Tarball文件具有.tar.gz、.tar.bz2或TGZ形式的文件扩展名，Tarball文件可用于装配源代码或二进制代码文件。在开源社区中，Tarball文件经常用于发行源代码，如果下载的文件名字包含一个.tar.gz后缀，首先需要双击文件项或图标，解压相应的文件，然后才能安装其中包含的软件。在命令行终端窗口，可以使用“tar -xzf filename”命令解压相应的文件，然后再执行其中包含的软件安装命令。

软件包也包含数字签名，以证实其来源是否真实有效。软件包管理工具利用GPG公钥验证软件包数字签名的合法性。

2. 软件仓库

软件仓库指的是一个网站或存储目录，其中提供按一定组织形式存储的软件包与索引文件。利用软件仓库，软件维护工具能够检索与获取正确的软件包，完成软件的安装，以及Linux系统的更新与升级。利用软件仓库，使用单个命令即可安装选定的软件包，更新所有的系统软件，或者找出匹配指定检索准则的软件包。

针对不同版本的Ubuntu Linux系统，Ubuntu项目组提供数千个可直接安装的程序。这些程序以软件包形式存储在不同的软件仓库中，供用户利用Internet下载与安装，使用户能够非常容易地安装新的软件或更新现有的程序。此外还提供诸如安装介质或映像，以及源码等。

基于Ubuntu项目组能够提供的软件支持程度，以及是否遵循Ubuntu的自由软件理念，Ubuntu的软件仓库主要分为四种类型（或渠道）：

- Main——Ubuntu官方完全支持的软件，也是Ubuntu Linux系统的基本软件包，能够构成一个完整的Linux系统。其中包含大多数流行的、稳定的、可以自由发布的开源自由软件。也是安装Ubuntu Linux系统时自动安装的软件。
- Restricted——Ubuntu支持的，但没有自由软件版权的通用软件。Ubuntu项目组无权直接修改此类软件，因而也无法提供完全的技术支持，即便如此，Ubuntu项目组还是会向软件作者反馈其中存在的问题。
- Universe——由Ubuntu社区维护，Ubuntu项目组不提供官方支持，只是为扩展与丰富Linux基本系统的功能与应用范围而提供的附加软件包。Universe软件仓库包含数以千计的软件，是整个自由、开源Linux世界的缩影，从中可以找到大多数开源软件。此类软件大都是基于Main中的组件编写的，因而能够与Main提供的软件相安无事地共同运行，只是在稳定性与安全升级方面缺乏保障。
- Multiverse——“非自由软件”，这意味着其中的软件并不满足自由软件的版权规定。使用此类软件时，如何处理版权及尊重版权所有者的劳动，完全依靠用户自己把握。这些软件通常较少修改和更新，需由用户自己承担所有风险。

Ubuntu的CD/DVD安装映像主要包含来自Main和Restricted两个仓库中的软件，但并不完整，尤其是CD，其中仅提供一小部分软件。在安装过程中，如果能够连网，可以在安装过程中补充安装诸如中文环境等更多的软件包。安装就绪之后，利用系统中已经安装的软件包管理工具，也可以直接检索、安装包括Universe或Multiverse在内的4个软件仓库中的任何软件。

用户应当总是使用软件仓库，以便确保能够收到最新版本的软件。如果同一软件包存在多个可用的更新版本，apt-get、aptitude或synaptic等软件维护工具能够自动选用最新版本的软件。

3. 软件包的相互依赖关系

一个软件包是一个相对独立的基本功能单元，但大多数软件包通常都需要一定的底层支持，如函数库或底层协议支持等，而一个函数库或底层协议可能会同时支持多个软件包。当一个软件包需要某个特定的函数库或底层协议支持时，包含函数库或协议支持的软件包就是当前软件包依赖的软件包。为了适当地安装一个软件包，必须首先满足其依赖关系。

apt-get等软件维护工具使用软件包的依赖关系数据，确保在安装或更新期间能够安装必要的软件包，自动安装系统中任何不存在的、与新装软件具有依赖关系的附加软件包，满足软件包的依赖关系。

13.2 利用apt-get管理软件包

APT (Advanced Package Tool) 是一个通用的综合软件管理与维护工具，功能完整，易于使用。apt-get、aptitude和synaptic等绝大多数软件包工具都是基于APT及其配置文件发展起来的，是APT的前端软件管理工具。

早期，APT的配置指令均存储在一个单独的配置文件/etc/apt/apt.conf中，Ubuntu Linux系统把这个文件分解成多个小型文件，存储在/etc/apt/apt.conf.d目录中。APT的cron脚本存储在/etc/cron.daily目录中，因此每天都会自动运行，读取/etc/apt/conf.d目录中的配置文件，根据其中的配置要求，维护APT本地软件包索引文件及APT本地缓存区。

/var/lib/apt/lists目录存有APT本地软件包索引文件。对于/etc/apt/sources.list文件中配置的每一个软件仓库，这个目录中均存在一个文件，其中列出了相应软件仓库中每一个软件包的最新版本信息。APT使用这些文件确定软件包是否已经安装到本地系统，哪些软件包位于本地缓冲目录，软件包的最新版本是什么等。

/var/cache/apt/archives目录是APT的本地缓冲目录，其中缓存了最近下载的deb软件包文件。通常，APT的cron脚本将会限制这个目录占用的存储空间及文件的存放时间。

除了/etc/apt/sources.list文件之外，APT还使用/etc/apt/apt.conf.d目录中的所有文件作为配置文件，其中包含各种APT配置参数。通常，用户只需关注sources.list配置文件，不必考虑其他文件中的配置参数，故这里不再赘述。

apt-get是一个命令行软件管理工具，能够利用软件仓库安装选定的软件包，或删除、更新系统中已安装的软件包，升级Linux系统。apt-get命令的语法格式简写如下：

```
apt-get [options] { update | upgrade | dist-upgrade | dselect-upgrade
| check | clean | autoclean | autoremove }
apt-get [options] { install | remove | purge } pkgs
```

apt-get命令的选项分为功能选项与普通意义的选项，功能选项表示一个具体的动作，如install表示安装，update表示更新，以及upgrade表示升级等。除了“-h”或“-help”等选项之外，apt-get命令要求用户必须提供一个功能选项。表13-1给出了apt-get命令支持的部分功能选项及简单说明。

表13-1 apt-get命令支持的部分功能选项

功能选项	简单说明
install <i>pkgs</i>	安装。用于安装指定的软件包。在指定软件包的名字时，只需给出其缩写形式，不必完整写出，例如，指定软件包libc6_1.9.6-2.deb时，只需指定libc6即可。同时，apt-get还会安装指定软件包依赖的所有底层支持软件包，以满足软件包的依赖关系。/etc/apt/sources.list文件用于指定期望的软件源。如果没有恰好匹配的软件包，则假定指定的软件包名是一个检索模式，apt-get将会安装匹配指定名字模式的任何软件包。如果软件包名字后面附加一个减号“-”（中间没有空格）后缀，且软件包已经安装，apt-get将会删除指定的软件包。类似的，如果软件包名字后面附有一个加号“+”后缀，表示安装指定的软件包。为了选择安装一个特定版本的软件包，可以在软件包名字后面附加一个“=version”后缀，如“aptitude install apt=0.3.1”。同样，为了从一个特定的发行中选择一个软件包，可在软件包名字后面附加一个“/distribution”或“/archive”后缀，如stable、testing或unstable等。软件包的名字也可以看做一个表达式，如果没有软件包能够匹配给定的表达式，且表达式中包含句点“.”、问号“?”或星号“*”等特殊字符之一，则意味着这是一个正则表达式，因而可以用之检索软件仓库中的所有软件包，然后安装（或删除）与之匹配的任何软件包。注意，所谓匹配指的是子串意义上的匹配，因此“lo.*”能够匹配“how-lo”和“lowest”。此外，除了上述三个特殊字符外，还可以在正则表达式中使用上箭头“^”或美元符号“\$”等
update	更新。用于同步软件源的软件包索引文件，从/etc/apt/sources.list文件中指定的软件源中获取可用软件包的索引。例如，当使用deb格式的软件包档案文件时，apt-get命令将会检索Packages.gz文件，从中获取可用的新软件包或更新软件包信息。因此，在利用upgrade或dist-upgrade功能选项升级整个系统之前，应首先利用update功能选项，更新可用软件包的索引
upgrade	升级。从/etc/apt/sources.list文件指定的软件源中，下载并安装比当前系统已安装版本更新的所有软件包，但不会删除系统中已安装的软件包，也不会下载与安装系统中尚未安装的软件包。在执行系统升级之前，首先必须执行update，更新软件包索引，以便apt-get能够知道是否存在可用的新版软件包
remove <i>pkgs</i>	删除。从系统中删除（卸载）指定的软件包，同时删除依赖于指定软件包的其他软件包。除了删除软件包，remove还等同于install功能选项。例如，如果指定的软件包名字后面附加一个加号“+”（中间没有空格），将会安装而不是删除指定的软件包
autoremove	自动删除。用于删除为满足依赖关系而自动安装的，且当前不再需要的软件包
purge <i>pkgs</i>	清除。除了彻底清除软件包提供的配置文件等之外，其功能等同于remove功能选项
source <i>pkgs</i>	用于下载最新版本的源代码软件包，然后置于当前目录。如果想要下载特定版本的源代码软件包，可以采用“pkgname=version”的形式指定软件包的名字与版本号
check	诊断。用以更新软件包缓存区，检测软件包的依赖关系是否存在问题
clean	清除。用于清除缓存在本地目录中的软件包文件等。除了位于/var/cache/apt/archives和/var/cache/apt/archives/partial目录中的封锁文件，clean功能选项将会清除软件包的任何文件。当以dselect方法运行APT软件包管理工具时，将会自动执行清除功能。在不采用dselect方法维护软件包时，应注意随时运行“apt-get clean”命令，以释放磁盘空间

(续表)

功能选项	简单说明
autoclean	类似于clean，autoclean也用于清除缓存在本地目录中的软件包文件等。其差别在于后者仅删除不再继续下载且基本上不再继续使用的软件包文件。这将防止缓存空间由于长期没有清空而导致的增长失控
check	诊断。用于更新软件包缓冲区，检测受损的软件包依赖关系
dist-upgrade	除了执行upgrade功能之外，dist-upgrade还可以智能地处理由新版软件包导致的依赖关系变化。apt-get具有一个“智能的”冲突解决机制，如果需要，它将会尝试优先升级最重要的软件包。/etc/apt/sources.list配置文件包含一系列软件源定义，使apt-get能够获取期望的软件包

表13-2 给出了apt-get命令支持的其他选项及简单说明。

表13-2 apt-get命令支持的其他选项

选项	GNU选项	简单说明
-h	-help	显示apt-get命令用法的简明说明信息
-c <i>file</i>	--config-file= <i>file</i>	指定apt-get命令使用的软件源配置文件，其中包含软件仓库的地址或路径（如http、ftp、cdrom或本地文件）
-d	--download-only	仅仅下载软件包，既不解压，也不安装软件包
-q	--quiet	安静模式。生成适合于日志的输出信息，忽略安装进度指示信息
-u	--show-upgraded	显示已升级的所有软件包列表
-y	--yes, --assume-yes	对需要用户确认是否（yes/no）的所有请求，总是使用yes作为回答。这意味着采用非交互式方式自动运行apt-get命令
	--allow-unauthenticated	无需考虑软件包是否已经认证。即使软件包未认证，也不输出任何提示信息
	--no-download	禁止下载软件包。最好与“-m”选项一同使用，强制APT完全使用已下载的、缓存在本地系统中的软件包，执行软件包的安装与更新
	--reinstall	对于当前已安装的软件包，重新安装最新版本软件包

在使用apt-get命令时，首先需要从表13-1中选择一个功能选项，然后再根据功能选项的要求，指定准备安装或删除的软件包。作为补充，必要时也可以选用表13-2中列举的其他选项。对于每一个处理请求，apt-get将会根据系统缓存的索引文件，从配置的软件仓库中检索最新的软件包信息，确定下一步需要采取的处理动作，其中包括为了满足软件包的依赖关系，是否需要安装、更新或删除其他软件包等。

13.2.1 安装软件包

install功能选项主要用于安装指定的软件包。在正式开始下载与安装指定的软件包及其相关软件包之前，apt-get将会提请用户予以确认。如果没有异议，可输入“y”，接着按Enter键表示确认，然后即可开始下载与安装。如果不同意，可以输入“n”接着按Enter键终止下载，从而终止安装指定的软件包。在安装过程中，apt-get首先会依次下载每一个软件包，接着一一解压，然后再替换之前安装的软件包，并执行安装后的设置，如启动后台守护进程等。

在安装指定的软件包时，如果其依赖的软件包尚未安装，apt-get将会尝试从软件仓库中检索、下载必要的软件包（如库函数或通用软件），以满足软件包的相互依赖关系。在此情况下，apt-get仍会提请用户予以确认。

例如，在利用CD/DVD介质安装Ubuntu Linux系统时，安装程序不会安装文件系统限额软件包quota。为了安装quota软件包，可以使用下列apt-get命令：

```
$ sudo apt-get install quota
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  libnet-ldap-perl portmap
The following NEW packages will be installed:
  quota
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0B/474kB of archives.
After this operation, 1384kB of additional disk space will be used.
Preconfiguring packages ...
Selecting previously deselected package quota.
(Reading database ... 139954 files and directories currently installed.)
Unpacking quota (from .../quota_3.17-3ubuntu1_i386.deb) ...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
Setting up quota (3.17-3ubuntu1) ...
.....
$
```

当因故需要重新安装某个软件包时，可以利用apt-get命令的“-reinstall”选项与install功能选项。例如，为了重新安装quota软件包，可以使用下列apt-get命令：

```
$ sudo apt-get --reinstall install quota
Reading package lists... Done
Building dependency tree
Reading state information... Done
0 upgraded, 0 newly installed, 1 reinstalled, 0 to remove and 0 not upgraded.
Need to get 0B/474kB of archives.
After this operation, 0B of additional disk space will be used.
Do you want to continue [Y/n]? y
.....
$
```

13.2.2 软件更新与系统升级

如果想要全面更新或升级当前的Ubuntu Linux系统，首先需要使用apt-get命令的update功能选项，同步软件源的软件包索引文件，获取最新版本的可用软件包信息。然后再利用upgrade功能选项下载与安装新版的软件包。因此，在利用apt-get命令升级整个Ubuntu Linux系统时，首先应运行下列apt-get命令，确保软件包索引文件是最新的：

```
$ sudo apt-get update
[sudo] password for gqxing:
```

```

Get:1 http://debian.nctu.edu.tw karmic Release.gpg [189B]
Get:2 http://debian.nctu.edu.tw karmic-updates Release.gpg [189B]
Get:3 http://debian.nctu.edu.tw karmic-security Release.gpg [189B]
Get:4 http://debian.nctu.edu.tw karmic Release [65.9kB]
Get:5 http://debian.nctu.edu.tw karmic-updates Release [44.1kB]
Get:6 http://debian.nctu.edu.tw karmic-security Release [44.1kB]
.....
Fetched 10.7MB in 2min 17s (77.8kB/s)
Reading package lists... Done
$

```

然后运行下列apt-get命令，下载、安装新版的软件包，从而升级整个系统：

```

$ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages have been kept back:
  linux-generic linux-headers-generic linux-image-generic
The following packages will be upgraded:
  gdm grub-common grub-pc ntpdate
4 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Need to get 2166kB of archives.
After this operation, 0B of additional disk space will be used.
Do you want to continue [Y/n]? y
.....
$

```

系统的升级实际上是一个软件包的删除与重装过程，作为软件包更新与系统升级过程的一部分，apt-get将会自动删除老的软件包。

对于内核软件而言，作为一个备份，更新之前的内核软件将会继续保持在系统中，以便在使用新的内核引导系统出现故障时，用户能够有机会选用先前的系统内核继续引导系统。

13.2.3 删除软件包

利用apt-get命令的remove与purge功能选项，可以删除指定的软件包。注意，remove与purge功能选项的主要作用都是删除指定的软件包，但两者之间的不同之处在于：如果软件包中含有配置文件，remove功能选项在删除软件包时会保留配置文件，因此可称做部分删除，而purge功能选项将会删除整个软件包，包括配置文件，故可称做彻底删除。

在开始删除软件包之前，apt-get首先会考察系统中是否存在依赖于指定软件包的相关软件包。如果存在，apt-get会列出需要同时删除的其他软件包，提请用户确认。例如，为了从系统中删除不再继续使用的软件包quota，可以使用下列apt-get命令：

```

$ sudo apt-get remove quota
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  quota
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.

```

```
After this operation, 1384kB disk space will be freed.  
Do you want to continue [Y/n]? y  
.....  
$
```

13.2.4 安装存储介质中的软件包

通常情况下，应尽量使用Ubuntu项目或本地镜像网站提供的软件仓库和标准的软件管理工具下载、安装及更新软件。此外，也可以采用CD/DVD安装介质或存储在本地系统的软件包档案文件，安装尚未安装的软件包。

在安装CD/DVD或存储载在本地系统中的软件包时，需要事先配置sources.list文件，参见下一节的讨论。

13.2.5 sources.list配置文件

1. 简介

由于apt-get、aptitude及synaptic等大部分软件包维护工具均以APT作为底层支撑，而APT又主要使用/etc/apt/sources.list配置文件定义软件的发行源，故这一节主要介绍sources.list配置文件。在安装了Ubuntu Linux系统之后，系统将会创建一个默认的配置文件，使APT能够根据sources.list配置文件的设置，利用Ubuntu项目或镜像网站提供的软件仓库，执行软件包的安装、更新与升级等。

除了sources.list主配置文件之外，也可以在/etc/apt/sources.list.d目录中创建部分辅助配置文件，作为sources.list配置文件的补充。在辅助配置文件中，定义软件源的语法格式及文件扩展名与sources.list主配置文件完全一样。

为了增加、修改软件仓库，或改变APT的默认处理动作，如增加本地软件仓库的镜像站点，通常需要修改/etc/apt/sources.list配置文件。在修改sources.list配置文件时，可以使用任何文本编辑器。

sources.list配置文件用于定义各种可用的软件源，如网站、镜像站点、CD/DVD以及本地存储目录等。在sources.list配置文件中，每行只能定义一个软件源。其语法格式如下：

```
type uri distribution [component1] [component2] [...]
```

其中，type表示软件包档案文件的类型，Ubuntu Linux系统支持的软件包档案文件类型主要有deb（二进制可执行代码）与deb-src（源代码）两种；uri用于定义软件源的URI地址类型，如http等。distribution可以是stable、unstable或testing等表示软件版本类型的关键字之一；每个component可以是main、contrib、non-free或非-us等表示软件仓库类型的关键字之一。

在定义deb类型的软件包时，必须在uri字段中指定软件源的主要发行版本（基准存储位置），以便APT能够从中找出其需要的信息。存储位置可以是一个路径名，如果路径名以斜杠字符“/”结束，必须省略后续的软件仓库定义部分。

主要发行版本的基准存储位置可以包含一个“\$(ARCH)”变量，以便APT能够在运行过程中替换成符合本地系统的机型，如i386、m68k或powerpc等。这种定义方式允许设定一个通用的sources.list配置文件。

在sources.list配置文件中，每行只能定义一个软件源，故对同一个URI而言，如果其中存在不同类型的软件包档案文件，可能需提供多个软件源定义。在读取所有的软件源定义之后，APT将会对所有URI进行排序，把相同的URI合并为对同一个主机的引用，建立单一的网络连接。APT也会并发地连接到不同的主机，以便能够有效地传输文件。

在sources.list配置文件中，定义软件源的优先顺序非常重要。通常应把最常用的、速度最快的软件源放在最前面，例如通常应把CD/DVD、本地主机以及速度快的本地镜像网站依次排列在前面。

sources.list配置文件当前支持的URI地址类型包括file、cdrom、http和ftp等，分别简述如下：

- file——表示可以使用文件系统中的任何目录作为软件包的软件源。这对于采用NFS或本地镜像方式安装Ubuntu Linux系统或软件包是非常用的。
- cdrom——表示使用CD/DVD作为软件包的软件源。
- http——用于指定一个HTTP服务器，其中提供软件包的档案文件。
- ftp——用于定义一个FTP服务器，其中提供软件包的档案文件。
- copy——除了把软件包复制到本地系统的缓存目录，而不是在原目录位置直接使用之外，copy完全等同于file。对于使用zip磁盘作为软件源的用户而言，这种地址类型是非常有用的。
- rsh/ssh——表示采用rsh/ssh方法连接远程主机，以预定用户的身份访问其中的软件包档案文件。采用此方式时，不能出现密码认证过程，因此，必须事先配置好RSA密钥或rhosts主机认证。在访问远程文件时，采用标准的find与dd命令，实现本地系统与远程主机之间的文件传输。

2. 示例

当本地系统（包括采用NFS安装的远程资源）存有可安装的软件包时，可以选用file作为URI地址类型，指定软件仓库的位置。例如，假定/opt/debian目录中存有稳定版本的（stable）发行软件，如stable/main、stable/contrib以及stable/non-free三种类型的软件包时，可以在sources.list文件中增加下列定义：

```
deb file:/opt/debian stable main contrib non-free
```

如果想用CD/DVD作为软件源，可以采用cdrom作为URI地址类型，指定软件存储的目录位置。以9.04版Ubuntu Linux系统的DVD安装介质为例，可以在sources.list文件中增加下列软件源定义（如果不知道怎样在文件中增加CD/DVD软件源定义，可以运行apt-cdrom命令）：

```
deb cdrom:[Ubuntu 9.04 _Jaunty Jackalope_ - Release i386 (20090421.3)]/ jaunty
main restricted
```

为了使用FTP协议，访问存储在ftp.debian.org网站debian目录中的stable/contrib类型的软件包档案文件，可以在sources.list文件中增加下列定义：

```
deb ftp://ftp.debian.org/debian stable contrib
```

若想使用HTTP协议，访问cn.archive.ubuntu.com网站ubuntu目录中的软件包档案文件，可以在sources.list文件中增加下列定义：

```
deb http://cn.archive.ubuntu.com/ubuntu/ jaunty main restricted
```

下面是在安装了Ubuntu 9.10 Linux系统，软件源改为中国台湾之后的sources.list配置文件：

```
$ cat /etc/apt/sources.list
# deb cdrom:[Ubuntu 9.10 _Karmic Koala_ - Release i386 (20091028.2)]/ karmic main
restricted
.....
deb http://tw.archive.ubuntu.com/ubuntu/ karmic main restricted
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic main restricted
.....
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-updates main restricted
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-updates main restricted
.....
deb http://tw.archive.ubuntu.com/ubuntu/ karmic universe
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic universe
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-updates universe
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-updates universe
.....
deb http://tw.archive.ubuntu.com/ubuntu/ karmic multiverse
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic multiverse
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-updates multiverse
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-updates multiverse
.....
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-security main restricted
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-security main restricted
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-security universe
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-security universe
deb http://tw.archive.ubuntu.com/ubuntu/ karmic-security multiverse
deb-src http://tw.archive.ubuntu.com/ubuntu/ karmic-security multiverse
$
```

13.3 利用aptitude管理软件包

在Ubuntu Linux系统中，aptitude是一个完全可以替代apt-get的软件管理工具，两个命令中的大多数功能选项与参数也完全兼容，可用于安装、查询或删除软件包，以及升级整个Ubuntu Linux系统。aptitude命令的语法格式简写如下：

```
aptitude [options] { update | safe-upgrade | clean | autoclean }
aptitude [options] { install | reinstall | full-upgrade | download
                    | remove | purge | show } pkgs
aptitude [options] search pkg-pattern
```

其中，pkgs为软件包档案文件的名字。在运行aptitude命令时，必须选择一个具体的动作：如install（安装）、update（更新）、safe-upgrade（升级）、show（查询）、remove（删除）以及search（检索）等，详见表13-3。考虑到实用性，这里主要介绍软件包的安装、查询、删除与检索，以及系统的升级等功能。

在aptitude命令的安装、升级及更新等功能选项中，还可以选用部分普通选项，以限定功能选项的处理动作，详见表13-4。

表13-3 aptitude命令的部分功能选项

功能选项	简单说明
<code>install pkgs</code>	<p>安装指定的软件包。如果指定软件包的名字中含有波浪号“~”，表示这是一个检索模式，意味着安装匹配检索模式的所有软件包。为了安装一个特定版本的软件包，可在软件包的名字后面附加一个“=version”后缀，如“<code>aptitude install apt=0.3.1</code>”。类似的，为了从一个特定档案文件中选择一个软件包，可在软件包的名字后面附加一个“/archive”后缀，如“<code>aptitude install apt/experimental</code>”。在aptitude命令中，如果在指定软件包的名字后面附加一个“强制修饰符”，可对相应的软件包执行不同的处理动作。例如，“<code>aptitude install quota-</code>”命令表示删除而非安装quota软件包。aptitude命令支持下列强制修饰符：</p> <ul style="list-style-type: none"> • <code><package>+</code>: 安装指定的软件包 • <code><package>+M</code>: 安装指定的软件包，并立即增加一个自动安装标记（注意，如果没有任何软件包依赖于指定的软件包，将会立即删除指定的软件包） • <code><package>-</code>: 删除指定的软件包 • <code><package>_</code>: 清除指定的软件包（删除指定的软件包，及有关的配置文件和数据文件） • <code><package>=</code>: 保持指定的软件包，取消任何安装、删除或升级动作，防止将来自动升级指定的软件包 • <code><package>=</code>: 保持当前版本的软件包不变，取消任何安装、删除或升级动作，但不能阻止将来的自动升级 • <code><package>+M</code>: 把指定的软件包标记为自动安装的软件包 • <code><package>+m</code>: 把指定的软件包标记为手工安装的软件包 <p>注意，一旦在最终确认时输入了“Y”，“<code>aptitude install</code>”命令将会修改aptitude保存的动作执行信息。因此，如果输入了“<code>aptitude install foo bar</code>”命令，一旦在下载和安装过程中间终止了软件包的安装，需要运行“<code>aptitude remove foo bar</code>”命令，取消前者造成的一致性问题的。</p>
<code>update</code>	从/etc/apt/sources.list配置文件定义的软件源中更新可用的软件包索引，其功能等价于“ <code>apt-get update</code> ”命令
<code>safe-upgrade</code>	对于系统中已安装的软件包，升级到最新版本。除非不再继续使用，不会删除原先已经安装的软件包。注意，选用这一功能选项时也不会安装之前尚未安装的软件包。与full-upgrade相比，safe-upgrade相对更保守一些，为了升级或安装更多的软件包，可以使用full-upgrade功能选项
<code>full-upgrade pkgs</code>	对于已安装的软件包，升级到最新版本。删除或安装必要的软件包。与safe-upgrade相比，full-upgrade功能选项有可能执行一些不需要的动作，但能升级或安装一些safe-install功能选项无法升级或安装的软件包
<code>reinstall pkgs</code>	重新安装指定的软件包
<code>remove pkgs</code>	删除指定的软件包
<code>purge pkgs</code>	彻底清除指定的软件包
<code>search pattern</code>	<p>检索匹配命令行指定模式的所有软件包。通常，“<code>aptitude search</code>”命令的输出形式如下所示：</p> <pre> i apt - Advanced front-end for dpkg pi apt-build - frontend to apt to build, optimize and in cp apt-file - APT package searching utility -- command- i Mysql-server-5.0 - MySQL database server binaries </pre> <p>在search功能选项的输出信息中，每行显示一个检索结果。其中，第一列中的第一个字符表示软件包的安装状态：最常见的安装状态是“p”，意味着系统中不存在相应的软件包</p>

（续表）

功能选项	简单说明
	记录；“c”表示相应的软件包已经删除，但其配置文件仍遗留在系统中；“i”表示相应的软件包已经安装；“v”表示虚拟软件包。如果存在，第二个字符表示软件包的存储动作，最常见的存储动作是“i”（意味着即将安装的软件包）、“d”（意味着即将删除的软件包）和“p”（意味着即将清除的软件包，包括其配置文件）。如果存在第三个字符“A”，表示相应的软件包将会自动安装（有关安装状态与动作标志的完整说明，可以查询Aptitude参考手册“Accessing Package Information”一节）
show pkgs	显示指定软件包的详细信息。如果指定软件包的名字中含有波浪号“~”，将会把软件包名看做检索模式，显示与之匹配的所有软件包信息。如果命令行中提供一个或多个“-v”选项，将会显示所有版本的软件包信息，如软件包适用的机型、压缩后的文件大小及文件名等。否则，仅显示“候选版本的软件包”，即“aptitude install”命令可能会下载的软件包信息。如果软件包名字后面附加一个“=version”后缀，可以显示指定版本的软件包信息。类似地，软件包名字后面附加一个“/archive”后缀，可以显示特定档案文件中的软件包信息
clean	从软件包缓存目录（/var/cache/apt/archives）中删除先前下载的所有软件包文件
autoclean	从软件包缓存目录（/var/cache/apt/archives）中删除不再继续下载的任何软件包文件。这将防止在长期没有清空缓存空间时导致其增长失控
download pkgs	下载指定的软件包文件，并存储在当前目录中。通常，aptitude会下载与“aptitude install”命令安装版本相同的软件包。如果在软件包名字后面附加一个“=version”后缀，可以下载指定版本的软件包。如果在软件包名字后面附加一个“/archive”后缀，还可以从指定档案文件中下载相应版本的软件包

表13-4 aptitude命令的部分普通选项

选项	GNU选项	简单说明
-h	--help	显示aptitude命令用法的简要说明
-d	--download-only	仅把软件包下载到本地缓存目录，而不执行任何安装或删除动作。软件包的默认缓存目录为/var/cache/apt/archives
-D	--show-deps	与安装（如install、full-upgrade）或删除（如remove、purge）等功能选项一起使用时，能够输出自动安装和删除的软件包简要说明信息
	--purge-unused	清除已安装的、任何软件包都不再需要的软件包
-v	--verbose	输出aptitude命令运行过程的说明信息，这个选项能够使部分功能选项（如show）显示更多的信息。如果同时提供多个“-v”选项，可以提供更多的信息
-y	--assume-yes	对需要用户确认（yes/no）的所有请求，总是使用yes作为回答。这也意味着采用非交互方式自动运行aptitude命令，执行软件包的安装、升级或删除
-Z		在安装、升级或删除的软件包时，显示每个软件包占用或释放的磁盘空间

13.3.1 安装软件包

install功能选项用于安装指定的软件包。因此，为了安装尚未安装的软件包，可以使用下列aptitude命令：

```
$ sudo aptitude install quota
```

如同apt-get命令一样, aptitude能够在安装软件包的同时, 自动解决软件包的相互依赖关系问题。如果准备安装的软件包依赖的软件包尚未安装, aptitude将会在提请用户确认之后同时下载并安装相关的软件包。

如果因故需要重新安装一个已安装的软件包, 如删除了其中的文件, 或欲恢复初始的配置文件, 可以使用aptitude命令的reinstall功能选项。例如, 使用下列apt-get命令, 可以重新安装quota软件包:

```
$ sudo aptitude reinstall quota
```

13.3.2 更新与升级

aptitude的升级功能基本上类似于apt-get命令。例如, 为了全面升级当前的Ubuntu Linux系统, 首先需要使用下列aptitude命令, 同步软件源的软件包索引文件, 获取最新的可用软件包版本信息:

```
$ sudo aptitude update
```

然后使用下列命令, 下载与安装所有需要更新的软件包, 从而升级整个系统:

```
$ sudo aptitude safe-upgrade
```

13.3.3 查询软件包

aptitude命令的show功能选项用于查询各种软件包信息。例如, 利用下列aptitude命令, 可以查询软件包的安装状态、版本、依赖关系、档案文件的大小及简单说明等信息:

```
$ aptitude show vsftpd
Package: vsftpd
State: installed
Automatically installed: no
Version: 2.2.0-1ubuntu1
Priority: extra
Section: net
Maintainer: Ubuntu Core Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Uncompressed Size: 479k
Depends: libc6 (>= 2.7), libcap2 (>= 2.10), libpam0g (>= 0.99.7.1), libssl0.9.8
(>= 0.9.8f-5), libwrap0 (>= 7.6-4~), debconf (>= 0.5) | debconf-2.0,
adduser, libpam-modules, netbase, lsb-base (>= 1.3-9ubuntu3), ssl-cert
(>= 1.0-11ubuntu1), sysv-rc (>= 2.86.ds1-14.1ubuntu2), update-inetd
Recommends: logrotate
Conflicts: ftp-server
Replaces: ftp-server
Provides: ftp-server
Description: lightweight, efficient FTP server written for security
This package provides the "Very Secure FTP Daemon", written from the ground up
with security in mind.

It supports both anonymous and non-anonymous FTP access, PAM authentication,
bandwidth limiting, and the Linux sendfile() facility.
Homepage: http://vsftpd.beasts.org/
$
```

13.3.4 检索软件包

在安装软件包时，可以事先利用aptitude命令的search功能选项，从系统缓存的软件包索引文件，或从/etc/apt/sources.list配置文件设定的软件仓库中检索可用的软件包，也可以检索系统中已安装的软件包。aptitude检索功能的语法格式简写如下：

```
aptitude search pkg-pattern
```

其中，pkg-pattern是软件包的名字或名字模式。在指定软件包的名字时，可以指定一个确切名字或部分名字，或软件包简述中的一个关键字，也可以是一个特殊的名字模式。实际上，在指定search功能选项的软件包检索模式时，aptitude将会按照包含原则进行检索，给出匹配检索模式的所有软件包。

软件包的检索结果取决于检索模式，如果检索结果为空，即aptitude命令没有输出任何信息，意味着没有匹配检索模式的软件包。

如果已经知道软件包的准确名字，可以按照名字查询具体的软件包。例如，为了查询gawk软件包的相关信息，可以使用下列aptitude命令：

```
$ aptitude search gawk
i   gawk          - GNU awk, a pattern scanning and processing
p   gawk-doc      - Documentation for GNU awk
$
```

在search功能选项的输出信息中，每行显示一个检索结果。其中，第一列表示软件包的安装与存储状态（参见表13-3），第二列是软件包的名字，第三列是软件包的简单说明。

aptitude命令search选项的检索功能非常强，除了能够按照指定的名字模式检索软件包之外，aptitude还根据软件包的检索特点，提供一些特殊检索模式。表13-5给出了Ubuntu Linux系统中可用的部分特殊检索模式。

表13-5 aptitude命令可用的部分特殊检索模式

特殊检索模式	简单说明
<code>~a action</code>	检索已经标记为指定安装动作的软件包。安装动作可以是install、upgrade、downgrade、remove、purge、hold（用于测试软件包是否需要保持固定不变）或keep（用于测试软件包是否需要保持不变）等关键字之一
<code>'pattern1 pattern2'</code>	检索同时匹配指定模式的软件包
<code>~A archive</code>	检索指定软件包类型（如stable、unstable或testing等）中的所有软件包
<code>~b</code>	检索依赖关系不满足的软件包
<code>~BdepType</code>	检索不满足指定依赖类型的软件包，依赖类型可以是depends、predepends、recommends、suggests、breaks、conflicts或replaces等关键字之一
<code>~Cpattern</code>	检索与指定软件包冲突的软件包
<code>~c</code>	检索已经删除但其配置文件等尚未完全清除的软件包
<code>~d description</code>	检索其描述信息匹配指定描述关键字的软件包
<code>~E</code>	检索基本系统软件包（其控制文件中含有“Essential: yes”控制语句）
<code>~i</code>	检索系统中已安装的软件包

(续表)

特殊检索模式	简单说明
<code>~m maintainer</code>	检索指定作者维护的软件包
<code>~n name [name]</code>	检索匹配指定名字模式的软件包。例如，大多数库函数软件包均匹配“ <code>~n name</code> (<code>^lib</code>)”检索模式
<code>~N</code>	检索新的软件包
<code>'pattern1 pattern2'</code>	检索匹配指定表达式 <code>pattern1</code> 、 <code>pattern2</code> 或同时匹配两个表达式的软件包
<code>~I</code>	显示所有的软件包
<code>~U</code>	检索系统中已经安装且需要升级的所有软件包
<code>~v</code>	检索虚拟软件包

1. 查询所有的软件包

利用下列`aptitude`命令，可以列出Ubuntu提供的（不管系统中是否已经安装）所有软件包，这将是一个长长的软件包列表：

```
$ aptitude search ~I
```

2. 查询软件仓库中可供更新的软件包

利用下列命令，可以列出软件仓库中能够用于更新当前系统的所有软件包（如果不经常更新，这也将是一个长长的软件包列表）：

```
$ aptitude search ~U
i   gdm                      - GNOME Display Manager
i   linux-generic            - Complete Generic Linux kernel
i   linux-headers-generic    - Generic Linux kernel headers
i   linux-image-generic      - Generic Linux kernel image
$
```

3. 查询系统中已安装的软件包

利用下列命令，可以列出系统中已安装的所有软件包（这将是一个长长的软件包列表）：

```
$ aptitude search ~i
```

4. 组合检索功能

利用逻辑与及逻辑或特殊检索模式，可以组合两个检索模式，检索具有特定意义的软件包。例如，为了检索系统中已安装的Apache服务器软件包，可以使用下列`aptitude`命令：

```
$ aptitude search '~i apache'
i   apache2                  - Apache HTTP Server metapackage
i A apache2-mpm-worker       - Apache HTTP Server - high speed threaded m
i A apache2-utils            - utility programs for web servers
i A apache2.2-common         - Apache HTTP Server common files
$
```

在上述输出信息中，第3～5行第一列中的“A”标志说明相应的软件包是自动安装的，如表13-3所示。

13.3.5 删除软件包

aptitude命令的remove或purge功能选项用于删除或彻底清除指定的软件包。当利用remove功能选项删除指定的软件包时，如果软件包中含有配置文件，配置文件将会继续保留在系统中。而purge功能选项能够彻底清除指定的软件包。当某个软件包不再需要时，可以根据具体情况，采用remove或purge功能选项，部分或全部删除指定的软件包。例如，采用下列aptitude命令将会删除quota软件包，但其配置文件将会遗留在系统中：

```
$ sudo aptitude remove quota
Reading package lists... Done
Building dependency tree
Reading state information... Done
Reading extended state information
Initializing package states... Done
The following packages will be REMOVED:
  quota
.....
$ aptitude search ~c
c   quota                                - implementation of the disk quota system
$
```

如同安装软件包一样，删除软件包也存在软件包依赖关系的问题。如果删除的软件包是其他软件包的底层支撑软件包，aptitude将会提请用户决定是否继续删除指定的软件包。

13.4 synaptic软件管理工具

synaptic软件包管理器是一种基于APT开发的高级图形界面软件管理工具，其中实现了apt-get命令行工具的所有功能，启动时能够自动连接与检索预先定义的软件仓库，可用于安装或删除选中的软件包，浏览或检索可用的软件包，升级整个系统，以及查询软件包的说明信息等。

运行synaptic时，可在GNOME桌面中选择“系统→系统管理→新立得软件包管理器”菜单，或在命令行界面中输入synaptic命令。synaptic软件包管理器的主界面如图13-1所示。



图13-1 “新立得软件包管理器”主界面

在“新立得软件包管理器”主界面中，左下角的“组别”、“状态”、“源自”、“自定义过滤器”以及“搜索结果”五个按钮分别用于选择软件分组，查询软件包的安装状态，查询与设置软件源，定义过滤器，以及查询新近执行的搜索结果等。

当单击不同的按钮时，左上角的窗口将会随之显示相应内容。初始进入synaptic主界面时，窗口中会显示“全部”软件包。

单击左上角窗口中的任何一个软件组，右上角的窗口将会显示相应软件组中包含的软件包，其中第一列复选框表示软件包的安装状态。第2~6列分别是软件源的标志、软件包的名字、当前已安装的软件包版本、最新版本，以及软件包的简单描述。如果第一列的复选框为空，表示相应的软件包尚未安装；如果复选框中涂有颜色，说明相应的软件包已经安装。

当单击右上角窗口中的任何一个软件包时，右下角的窗口中将会给出软件包的更多说明信息。此外，选中右上角窗口中的某个软件包，单击工具栏中的“属性”按钮，或右击右上角窗口中的任何软件包，将会弹出一个菜单，单击“属性”菜单项也会给出相应软件包的说明信息，如图13-2所示。



图13-2 软件包属性对话框

在synaptic图形界面中，除了使用鼠标选择菜单与按钮之外，还可以使用快捷键，实现软件包的选择、标记，以及软件包的安装、删除与升级等，如表13-6所示。

表13-6 synaptic支持的快捷键

快捷键	简单说明
Ctrl-R	更新软件包列表
Ctrl-F	打开软件包搜索对话框
Ctrl-O	显示选定软件包的属性对话框
Ctrl-I	标记选定的软件包以便安装
Ctrl-U	标记选定的软件包以便升级
Delete	标记选定的软件包以便删除
Shift-Delete	标记选定的软件包以便彻底删除
Ctrl-N	取消全部标记
Ctrl-G	标记所有更新
Ctrl-E	强制安装某个版本的软件包
Ctrl-Z	撤销最近一次标记的变动
Ctrl-Shift-Z	重做最近一次变动
Ctrl-P	令标记的所有变动生效
Ctrl-Q	退出新立得软件包管理器

13.4.1 浏览软件包

在synaptic软件包管理器的主界面中，单击左下角的相应按钮，可以按照软件源、软件分组、软件包安装状态、自定义过滤器或最近的检索结果浏览可用的软件包。如果需要，用户也可以创建自己的过滤器。为了按照名字或说明信息检索软件包，可以在工具栏的“快速搜索”文本框中输入检索关键字，或选择“编辑→搜索”菜单，synaptic将会弹出一个“查找”对话框，如图13-3所示。

此时，可以在“搜索”下拉列表框中输入检索关键字，在“搜索位置”选择框中选择检索原则，如软件包名称、软件包描述和名称、维护者、版本、依赖关系以及软件包的提供者，然后执行准确或模糊检索，搜索结果如图13-4所示。



图13-3 软件包检索对话框



图13-4 软件包搜索结果

在“新立得软件包管理器”主界面右下角的窗口中，可以考察软件包的详细说明信息，如软件包的大小、依赖关系、建议安装的附加软件包，以及软件包的简单说明等。

13.4.2 安装软件包

在安装软件包时，可以首先单击“刷新”按钮（或按Ctrl-R组合键），以便synaptic再次联系软件源，获取最新的软件包信息。然后在左上角的窗口中选择软件分组，在右上角的窗口中选择软件包，双击软件包左边的复选框（或按Ctrl-I组合键），也可以右击选定的软件包，从弹出的上下文菜单中选择“标记以便安装”菜单项，如图13-5所示。

如果选定的软件包要求安装其他相关的软件包，synaptic还会弹出一个对话框，列出尚需安装的底层支撑软件包。单击“标记”按钮，表示同意安装附加的软件包，如图13-6所示。

选中软件包及其底层支撑软件包之后，复选框中将会出现一个弧形箭头，如图13-7所示。

一旦选定了所有软件包，单击主界面工具栏中的“应用”按钮（或按Ctrl-P组合键），弹出一个对话框，其中给出了即将执行的软件变动概要说明，如图13-8所示。

单击“应用”按钮，表示同意安装本次选定的所有软件包，即可开始安装选定的所有软件包。

如果选定的软件包与系统中已安装的软件包有冲突，synaptic还会给出一个警告信息。在此情况下，对话框中将会给出需要删除的软件包。如果无法肯定将要删除的软件包是否不再需要，在单击“应用”按钮之前，应仔细检查其功能和用途。



图13-5 选择欲安装的软件包



图13-6 尚需安装的底层支撑软件包

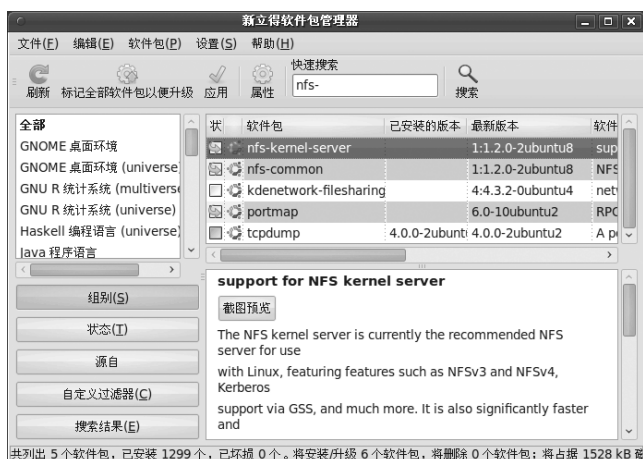


图13-7 选定待安装的软件包



图13-8 软件包安装汇总信息

13.4.3 删除软件包

若想删除或完全清除一个软件包，可在主界面右上角的窗口中右击选定的软件包，从弹出的上下文菜单中选择“标记以便删除”菜单项。如果选择“标记以便彻底删除”选项，则意味着删除选定的软件包及其有关的任何配置文件，如图13-9所示。

如果选定的软件包与系统中已安装的软件包存在依赖关系，synaptic将会给出一个警告信息，说明尚需连带删除的其他软件包。如无异议，可单击“标记”按钮，表示同意删除有关的软件包，如图13-10所示。

选中软件包以及影响其正常运行的其他软件包之后，复选框中将会出现一个红叉“×”，表示准备删除的所有软件包，如图13-11所示。

在标记了准备删除的所有软件包之后，单击主界面工具栏中的“应用”按钮（或按Ctrl-P组合键），系统将会弹出一个对话框，其中包含了选择结果的汇总信息。如果没有疑问，可单击“应用”按钮予以确认，如图13-12所示。



图13-9 选择欲删除的软件包



图13-10 删除软件包影响的其他软件包

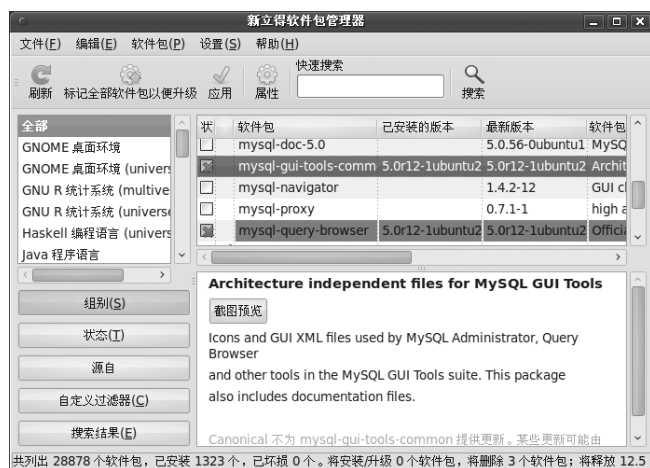


图13-11 选定待删除的软件包

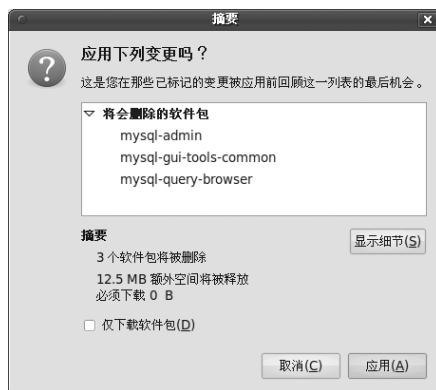


图13-12 软件包删除汇总信息

13.4.4 软件升级

在开始升级之前，可以单击主界面工具栏中的“刷新”按钮（或按Ctrl-R组合键），以便synaptic连接软件源，获取最新的软件包信息。如果安装的软件包需要更新，其左边的黄绿色复选框右上角将会出现一个五星标记。在升级软件时，可以按照软件包的分组，双击选定的软件包，或右击选定的软件包，从弹出的上下菜单中选择“标记以便升级”菜单项（或按Ctrl-U组合键），如图13-13所示。

如果选定软件包涉及的底层支持软件包需要一同安装与更新，系统还会弹出一个确认对话框，如图13-14所示。

选中软件包以及需要安装与更新的其他软件包之后，复选框中将会出现一个上升状的弧形箭头，如图13-15所示。

如果没有异议，单击主界面工具栏中的“应用”按钮（或按Ctrl-P组合键）予以确认。synaptic将会弹出一个对话框，其中包含软件包选择结果的汇总信息，如图13-16所示。

单击“应用”按钮确认之后，synaptic将会按照用户的选择，更新系统中已安装的软件包。



图13-13 选择欲升级的软件包



图13-14 安装与升级确认对话框



图13-15 选定待升级的软件包

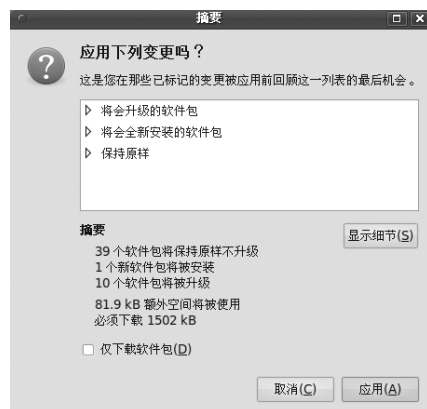


图13-16 软件包升级汇总信息

13.5 Ubuntu软件中心

Ubuntu软件中心是一种针对GNOME桌面菜单的软件管理工具，可以根据软件的功能或菜单分类，补充安装或删除选定的软件包，也可用于浏览、检索及查询可用的或已安装的软件包及其说明信息。

如果想要访问Ubuntu软件中心，可以在GNOME桌面中选择“应用程序→Ubuntu软件中心”菜单，连接到预定义的软件仓库，检索可用的软件包，最终出现如图13-17所示的“Ubuntu软件中心”主界面。

在“Ubuntu软件中心”主界面中，右侧窗口按照软件的功能或用途分类列出了若干软件分组。单击不同的软件组项，将会列出其中包含的一系列软件包，如图13-18所示。

通常，Ubuntu软件中心会分类列出所有可用的自由软件，如果仅想查询或选用Canonical公司支持的软件，可以单击“视图”菜单，从中选择“Canonical维护的应用程序”菜单项。

在软件包列表窗口中，可以利用右侧的滑块浏览软件包，也可以逐一输入软件包名字的部分英文字符，快速定位一个软件包。



图13-17 “Ubuntu软件中心”主界面

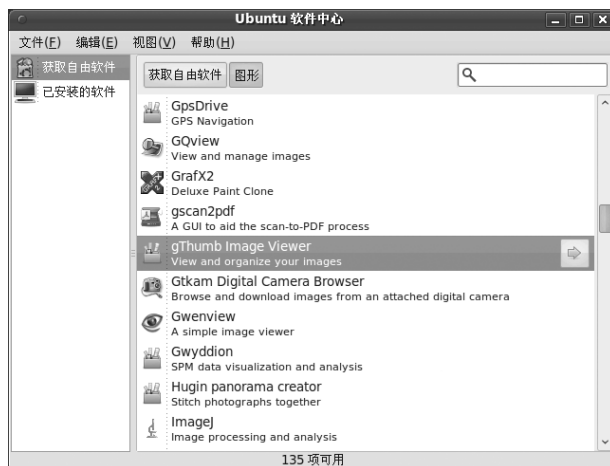


图13-18 软件包列表

主界面右上角的搜索框主要用于检索软件包。如果知道软件包的名字，可以采用其中的部分英文字符（如“gthumb”）作为关键字检索软件包。

选定软件包之后，双击软件包，或单击软件包后面的右箭头按钮，将会刷新窗口，显示软件包的简要说明，以及安装或移除按钮（取决于软件包是否已经安装）。此时可以单击“网站”按钮，访问相关的网站，以便了解更多的信息。也可以单击“安装（或移除）”按钮，安装（或删除）选中的软件包，如图13-19所示。

在输入sudo密码之后，系统将会刷新窗口，显示软件包的安装（或删除）进度，直至完成软件包的安装（或删除）。



Ubuntu软件中心能够安装的软件仅限于GNOME桌面菜单的软件组成部分，并非Ubuntu提供的全部软件包。对于其他软件而言，即使存在尚未安装的软件包，也无法利用Ubuntu软件中心安装（或删除）。例如，如果想要安装MySQL、Apache或DNS服务器等软件包，只能采用其他软件维护工具，如apt-get、aptitude或synaptic等。

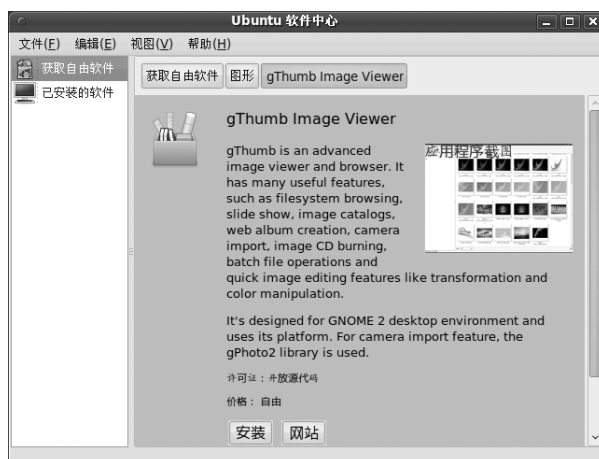


图13-19 软件包详细信息

13.6 软件包的更新

作为一个软件更新工具，Ubuntu Linux系统提供的update-notifier程序与update-manager脚本主要用于检测系统配置的软件仓库中是否存在可用的软件包。如果存在更新版本的软件包，GNOME窗口面板中将会出现一个后台运行的“更新管理器”窗口，提醒用户更新自己的系统。单击该窗口标题即可展开“更新管理器”窗口，如图13-20所示（此外，也可以在GNOME桌面中选择“系统→系统管理→更新管理器”菜单，打开“更新管理器”窗口）。



图13-20 “更新管理器”窗口

“更新管理器”窗口中列出了需要更新的所有软件包。第一列是复选框，其中的对钩“√”表示选定了相应的软件包；第二列是软件包的名字和简单说明。通常可以直接单击窗口左下角的“安装更新”按钮，在随后弹出的对话框中输入sudo密码，即可开始下载及安装软件包。之

后就是观察软件包的下载与安装进度，等待软件包安装结束，如图13-21所示。



图13-21 下载软件包

如果长时间没有更新系统，也可以先单击“检查”按钮，再次检索是否还存在其他需要更新的软件包。然后再单击“安装更新”按钮。

当需要更新的软件包较多，网速又比较慢时，最好按照软件包的重要程度，分批更新软件包。不然的话，一旦网络中断或需中途退出“更新管理器”窗口，将会前功尽弃，之前下载的软件包也无法再用。为此，可以单击部分软件包左侧的复选框，清除其中的对钩“✓”，暂不安装相应的软件包。

update-manager是一个简单易用的软件维护工具，也是Ubuntu Linux基本系统的一部分，每天都会以后台进程的形式自动运行，执行全面的系统更新检查，帮助用户实现系统软件的及时更新。

软件更新结束之后，如果需要重启系统，将会显示一个对话框，询问用户现在是否需要重新启动系统。此时可以酌情处理，如立即重启系统，或选择在之后的方便时间再重新引导系统。

第14章 文件系统管理

本章主要以Ext 2/3/4 文件系统为例，介绍文件系统的管理，详细讨论怎样创建和调整文件系统，怎样安装和卸载文件系统，怎样检测与修复受损的文件系统，以确保系统能够正常启动与运行，最后介绍文件系统的调试工具。

当加装新的磁盘等存储设备时，首先需要确保系统能够识别新增的设备，自动生成相应的设备文件。在能够以文件系统形式使用新的磁盘等存储设备之前，通常需要执行下列步骤：

(1) 首先利用fdisk命令（类似的命令还有cfdisk、sfdisk、parted以及partman等）对磁盘等存储设备进行分区。分区时，可以把整个磁盘划分为一个Linux分区。如果磁盘容量较大，或为了支持不同的应用，也可以把磁盘划分为若干分区，以便在每个分区中创建一个单独的文件系统，然后再把每个文件系统安装到Linux系统中。

(2) 利用mkfs或mke2fs等命令，在指定的磁盘分区中创建文件系统。

(3) 利用e2label命令，为新建的文件系统增加卷标（这一步是选用的）；

(4) 利用tune2fs命令，对文件系统的参数进行必要的调整（仅当需要时）；

(5) 利用mount等命令把新建的文件系统安装到指定的目录位置。

14.1 划分磁盘分区

每个物理磁盘可以划分为一个或多个逻辑磁盘，逻辑磁盘通称磁盘分区。磁盘分区信息位于磁盘扇区0的分区表中。在磁盘等存储设备上创建文件系统之前，首先必须对物理磁盘进行分区。在Linux系统中，一个磁盘分区既可用做文件系统，也可用做交换空间。

在Ubuntu Linux系统中，所有磁盘设备（包括USB移动硬盘）均采用/dev/sdDN的形式统一命名。其中，D可以是字母a、b、……分别表示第一个磁盘、第二个磁盘，依次类推。数字N从1开始编号，表示磁盘设备中的磁盘分区。例如，/dev/sda1是系统配置的第一个磁盘设备中的第一个磁盘分区。

通常，每个Linux系统至少需要两个磁盘分区，分别用做“/”文件系统和交换分区。也可以创建第三个磁盘分区，用做/boot文件系统，存储系统内核映像，以及引导系统时需要用到的其他辅助文件。在一个Intel x86 系列及其兼容的计算机系统中，用于引导系统的BIOS通常只能访问磁盘的前1024个柱面。因此，划分磁盘分区时需要确保BIOS能够访问到/boot分区。如果系统配置的磁盘容量较大，且出于安全、管理、备份或检测等原因，也可以把磁盘设备划分为多个分区，创建多个文件系统。

fdisk命令可用于划分磁盘设备，显示磁盘分区信息。在划分磁盘分区时，通常均采用交互方式，运行fdisk命令。fdisk命令的语法格式简写如下：

```
fdisk [-u] [-b sectorsize] [-C cyls] [-H heads] [-S sects] device
fdisk -l [-u] [device]
fdisk -s partition
```

其中，“-l”选项用于显示指定磁盘设备的分区表。如果未给定磁盘设备，则输出/etc/fstab文件中列举的，以及系统检测到的每个存储设备的分区表。“-u”选项表示以扇区（而不是以柱面）为单位列出每个设备分区的起始数据块位置。“-s”选项表示以数据块为单位显示指定设备分区的容量。

下面的例子说明了怎样利用fdisk命令，把64MB的U盘划分为一个Linux分区，以便创建文件系统（由于U盘容量通常比较小，没有必要划分多个分区。实际上，也可以不分区，直接在整个U盘上创建文件系统）。

```
$ sudo fdisk /dev/sdb

Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1015, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-1015, default 1015):
Using default value 1015

Command (m for help): p

Disk /dev/sdb: 65 MB, 65536000 bytes
3 heads, 42 sectors/track, 1015 cylinders
Units = cylinders of 126 * 512 = 64512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks    Id  System
/dev/sdb1              1         1015        63924    83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
$
```

在上述操作过程中，对磁盘分区表的任何改变，都是在内存中进行的。为了使磁盘分区信息生效，在退出fdisk交互命令之前，必须使用w（write）子命令写入磁盘。如果需要，还可以使用t（type）子命令设置磁盘分区的类型。如果不知道磁盘分区的类型代码，可以使用l（list）子命令显示磁盘分区类型代码表。

如果磁盘已经分区，并分别创建了不同的文件系统，为了获取磁盘分区信息，可以使用下列fdisk命令：

```
$ sudo fdisk -l /dev/sda

Disk /dev/sda: 40.0 GB, 40000536576 bytes
255 heads, 63 sectors/track, 4863 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x64cc19d0

   Device Boot      Start         End      Blocks    Id  System
/dev/sda1  *              1         1824       14651248+    7  HPFS/NTFS
```



```
/dev/sda2          1825          4731      23350477+   83   Linux
/dev/sda3          4732          4862      1052257+   82   Linux swap / Solaris
$
```

从上述输出信息中可知，磁盘分区1的数据块总数为14 651 248，其文件系统为NTFS。磁盘分区2和3的数据块总数分别为23 350 477和1 052 257，其中前者为Ubuntu Linux系统的“/”文件系统，后者为交换分区。

14.2 创建文件系统

在安装操作系统时，安装程序通常会引导用户提供必要的信息，或做出相应的选择，然后自动划分磁盘分区，创建文件系统，详见第1章“系统概述与安装”。在日常的实际应用过程中，仅当增加新盘，使用移动硬盘，或改变现有磁盘分区结构时，才有可能需要自己手动创建文件系统。

无论如何，在磁盘等存储设备分区之后，即可着手创建文件系统。创建文件系统时，主要有两种方法：一是使用最基本的、通用的mkfs命令，在选定的磁盘分区中创建指定的文件系统；二是利用各种特定的工具，如mke2fs、mkfs.ext2、mkfs.ext3或mkfs.ext4等，在选定的磁盘分区上直接创建指定类型的文件系统。

14.2.1 mkfs与mke2fs命令介绍

创建文件系统时，首先想到的通常是mkfs命令。在Linux系统中，mkfs命令实际上只是各种文件系统创建程序的一个总控程序，根据指定的文件系统类型，mkfs将会调用特定文件系统的创建程序“mkfs.fs”，其中fs可以是ext2、ext3或ext4等。因此，在创建一个具体的文件系统时，可以使用特定的文件系统创建命令。例如，为了创建一个Ext2/3/4文件系统，可以直接使用mkfs.ext2、mkfs.ext3或mkfs.ext4等命令，也可以使用等价的mke2fs命令。对于Ext2/3/4文件系统而言，mke2fs命令是最基本的工具。

用于创建Ext2/3/4文件系统的mkfs与mke2fs命令的语法格式简写如下：

```
mkfs [-V] [-t fstype] [fs-options] device [blocks]
mke2fs [-b block-size] [-g blocks-per-group] [-i bytes-per-inode]
[-I inode-size] [-j] [-J journal-options] [-L volume-label]
[-m reserved-blocks-percentage] [-n] [-N number-of-inodes]
[-S] [-t fstype] [-V] device [blocks]
```

表14-1给出了mkfs和mke2fs命令的选项及简单说明。

表14-1 mkfs/mke2fs命令的选项

选项	简单说明
-t fstype	指定创建的文件系统类型，如ext2、ext3或ext4等。如果未指定这个选项，默认文件系统的类型取决于存储设备的容量等因素
-b block-size	指定文件系统逻辑数据块的大小（字节数）。有效的逻辑数据块是1024、2048或4096个字节。如果未指定这个选项，mke2fs将根据整个文件系统的大小及预期的用途综合考虑而定

选项	简单说明
<code>-g blocks-per-group</code>	指定一个数据块组包含的逻辑数据块数量
<code>-i bytes-per-inode</code>	指定字节数与信息节点之比率，也即每个文件平均占用多少个字节数。这个数值反映了一个文件系统中每个文件的平均大小。数值越大，文件系统的信息节点数越少。这个数值通常不应小于文件系统逻辑数据块的大小。注意，文件系统一经创建，信息节点的数量是不能随意变动的，故应仔细地确定这个数值
<code>-I inode-size</code>	指定信息节点的大小（字节数），其默认值为128个字节
<code>-j</code>	创建一个具有Ext3日志功能的文件系统
<code>-J journal-options</code>	在创建Ext2文件系统时，使用指定的日志选项创建Ext3日志。多个日志选项中间可以加逗号“,”分隔符。实际上，创建一个具有日志功能的Ext2文件系统，不如直接创建一个标准的Ext3文件系统
<code>-L volume-label</code>	设置文件系统的卷标
<code>-m reserved-blocks-percentage</code>	指定创建文件系统时需要为超级用户或系统服务进程保留的数据块占整个文件系统容量的百分比。当达到这个数值时，只有超级用户才能继续请求分配空间。采用这一措施的目的是防止用户进程过多地占用存储空间而导致文件系统瘫痪，使超级用户能够在文件系统处于满负荷的情况下维护文件系统，或系统服务进程能够继续运行。这个选项的默认值是5%
<code>-n</code>	使用这个选项的目的只是为了让mke2fs显示，一旦需要创建文件系统时，最终将会创建一个什么样的文件系统，实际上并不会真正地创建文件系统。利用这个选项可以确定一个特定文件系统备份超级块的存储位置。当然，还要保证除“-n”选项之外，其他所有的选项与初次使用的命令选项是完全一致的
<code>-N number-of-inodes</code>	这个选项允许用户在创建文件系统时直接指定信息节点的数量，而不是采用mke2fs命令计算的信息节点默认值
<code>-S</code>	仅仅写入超级块和数据块组描述信息，而不是真正地创建文件系统。当所有的超级块和超级块备份均遭损坏，这个选项是非常有用的。如果所有的文件系统修复方法都不奏效的话，这是恢复文件系统的最后一招。使用这个选项时，mke2fs将会重新初始化超级块和数据块组描述部分，但不会触及信息节点区，以及数据块和信息节点位图。之后，应立即运行e2fsck程序。注意，这种方法并不保证能够挽救所有的数据
<code>-V</code>	用于显示mkfs或mke2fs命令的版本号
<code>-v</code>	在运行过程中输出更多的处理信息，如新建文件系统的类型等
<code>device</code>	设备文件名，表示准备创建文件系统的磁盘分区，如/dev/sdb1等
<code>blocks</code>	指定文件系统中数据块的总和。注意，这个值应等于准备创建文件系统的整个磁盘分区的大小。如果忽略，可由mkfs或mke2fs命令根据自动计算的结果确定整个文件系统的容量

利用“-n”选项并不实际创建文件系统的特性，可据此获取创建文件系统时mkfs命令使用的默认参数值，了解备份超级块的位置，以便当文件系统损坏严重时，可以利用备份超级块修复文件系统（当然，这样做的前提条件是除“-n”选项之外，其他所有的选项与初次使用mkfs命令时提供的选项必须是完全一致的）。示例如下：

```
$ sudo mkfs -t ext4 -n /dev/sdb1
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
```

```

Block size=1024 (log=0)
Fragment size=1024 (log=0)
16000 inodes, 63924 blocks
3196 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=65536000
8 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345

$

```

由上述命令的输出信息可以看出，除了位于数据块1的主超级块之外，其他备份超级块分别位于8193、24577、40961和57345四个数据块中。当文件系统的主超级块严重损坏时，可以利用任何一个备份超级块修复文件系统。

14.2.2 创建Ext2/3/4文件系统

下面的例子说明了怎样利用mkfs命令，在未分区的64MB U盘上创建一个Ext4类型的文件系统。其中，由于没有明确指定，文件系统逻辑数据块的大小采用默认值1024字节，整个文件系统的最大容量为63924个数据块，拥有16000个信息节点，可以创建或存储16000个文件。其中，文件系统保留了3196个数据块（5%）的存储空间，供超级用户在紧急情况下使用。文件系统分为8个数据块组，每个数据块组拥有8192个数据块，支持2000个信息节点。除了位于数据块1的主超级块之外，其他4个备份的超级块分别位于8193、24577、40961和57345号数据块中。执行mkfs命令之后，创建的文件系统具有一个根目录和一个lost+found子目录。

```

$ sudo mkfs -t ext4 /dev/sdb1
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
16000 inodes, 63924 blocks
3196 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=65536000
8 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 39 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

$

```

最后还要说明的是，在使用mkfs命令创建文件系统时，如果未用“-L”选项指定文件系统的卷标，可在事后采用e2label命令设定文件系统的卷标。e2label命令的主要功能就是显示和命名未安装文件系统的卷标，其语法格式如下：

```
e2label device [new-label]
```

其中，device是磁盘分区的设备文件名，如/dev/sda2等。new-label表示文件系统的卷标。如果未指定new-label，e2label命令将会输出文件系统当前的卷标。注意，在指定文件系统的卷标时，不能超过16个字符，例如：

```
$ sudo e2label /dev/sdb1
$ sudo e2label /dev/sdb1 data01
$ sudo e2label /dev/sdb1
data01
$
```

14.3 调整文件系统

一经创建，文件系统的基本组织结构原则上是不能随意变动的。如果需要，可以利用tune2fs命令，简单调整Ext2/3/4文件系统的部分可调参数，如在Ext2文件系统中增加日志功能，增加文件系统的卷标，增加索引，以提高大型目录的文件检索速度等；也可以实现文件系统的迁移，如把Ext2文件系统转换为Ext3文件系统，把Ext3文件系统转换为Ext4文件系统等；此外，还显示文件系统超级块中的重要信息。tune2fs命令的语法格式简写如下：

```
tune2fs [-l] [-c max-mount-count] [-C mount-count] [-e error-behavior] [-f]
[-u user] [-g group] [-i check-interval] [-j] [-J journal-options]
[-L volume-name] [-m reserved-blks-percentage] [-M last-mounted-dir]
[-O [^]features] [-r reserved-blks-count] [-T time-last-checked] device
```

表14-2 给出了tune2fs命令的部分选项及简单说明。

表14-2 tune2fs命令的部分选项

选项	简单说明
-c max-mount-count	调整文件系统的最大安装次数。超过此限，则强制系统采用e2fsck命令检测文件系统。如果把最大安装次数设定为0或-1，e2fsck命令与Linux系统内核将会忽略文件系统的安装次数。在使用具有日志功能的文件系统时，合理地设置文件系统的最大安装次数将会避免Linux系统同时检测所有的文件系统
-C mount-count	设置文件系统已经安装的次数。如果设置的安装次数大于最大安装次数（“-c”选项），在下次引导系统时，Linux系统将会使用e2fsck命令检测文件系统
-e error-behavior	指定当检测到文件系统出现错误时Linux内核应当采取的处理动作。文件系统的错误有可能导致Linux系统采用下列三种处理方式之一： · continue: 继续正常运行 · remount-ro: 以只读方式重新安装文件系统 · panic: 引起系统内核按系统瘫痪处理 在任何情况下，文件系统的错误都会促使Linux内核在下次引导系统时利用e2fsck命令检测文件系统

(续表)

选项	简单说明
-f	强制运行tune2fs命令, 即使操作过程中出现问题
-g <i>group</i>	设置能够使用文件系统保留数据块的用户组成员。给定的用户组参数既可以是一个数值形式的用户组ID, 也可以是用户组名。不管采用哪一种指定方式, 系统最终都会以数值形式把用户组ID写入超级块中
-i <i>check-interval</i> [d m w]	设定检测文件系统的最大时间间隔。如果指定的数值后面没有后缀, 或后缀为d, 则表示天数。后缀为m表示月数, 后缀为w表示周数。如果指定的数值为0, 表示禁止使用与时间有关的文件系统检测。Linux系统强烈建议用户启用“-c”或“-i”选项表示的文件系统检测功能, 强制系统周期地采用e2fsck命令执行全面的文件系统检测。否则有可能因坏块或内核漏洞等原因导致文件系统损害而长期毫无察觉, 直至最终引起数据的丢失或文件系统的严重受损
-j	在Ext2文件系统中增加Ext3日志功能。如果未指定“-J”选项, 采用默认的日志参数创建适当大小的日志
-J <i>journal-options</i>	<p>替换默认的Ext3日志选项size与device。日志选项采用等号“=”形式赋值。利用“-J”选项能够同时指定多个日志选项, 选项之间需加逗号“,”分隔符。Ext3文件系统支持下列日志选项:</p> <ul style="list-style-type: none"> • size=<i>journal-size</i>: 在文件系统中创建指定大小(单位MB)的日志。日志至少必须具有1024个逻辑数据块。也就是说, 如果逻辑数据块为1 KB, 则日志至少必须设为1 MB; 如果逻辑数据块为4 KB, 则日志至少必须设为4 MB。但最大不能超过102400个逻辑数据块 • device=<i>external-journal</i>: 利用外部日志设备实现文件系统的日志功能(注意, 外部日志设备与当前的文件系统必须具有相同大小的逻辑数据块)。在设置这个日志选项之前, 必须首先使用下列命令设定外部日志设备: <pre>mke2fs -O journal_dev external-journal</pre>
-l	输出文件系统超级块中的数据内容
-L <i>volume-label</i>	设置文件系统的卷标。Ext2/3/4文件系统的卷标可以长达16个字符。如果给定的卷标超过16个字符, tune2fs命令将会截掉超常部分, 并输出一个警告信息。在mount和fsck命令, 以及/etc/fstab文件中, 可以采用“LABEL=卷标”的形式, 以卷标替代设备名
-m <i>reserved-blks-percentage</i>	设置文件系统保留数据块的百分比
-M <i>last-mounted-dir</i>	设置文件系统最后一次安装的目录
-O [<i>^features</i>	<p>设置或清除文件系统的功能特性选项。如果在某个特性选项前加一个上箭头字符“^”, 表示清除相应的功能特性选项。如果特性选项前存在一个加号字符“+”或没有任何前缀, 则表示增加相应的功能特性。多个选项之间需加逗号“,”分隔符。tune2fs命令能够设置或清除的部分文件系统功能特性选项如下:</p> <ul style="list-style-type: none"> • dir_index: 采用散列B树提高大型目录的文件检索速度 • filetype: 在目录项中存储文件类型信息 • has_journal: 使用日志确保文件系统的一致性。设置这个文件系统功能特性等价于使用“-j”选项 • sparse_super: 限制备份超级块的数量, 以节省大型文件系统的空间 <p>在设置或清除文件系统的sparse_super与filetype功能特性之后, 必须运行e2fsck, 使文件系统保持一致性。如果需要, tune2fs将会给出一个消息, 要求系统管理员运行e2fsck命令。在设置文件系统的dir_index功能特性之后, 可以运行“e2fsck-D”命令, 把现有的目录转换成散列B树格式的目录</p>

（续表）

选项	简单说明
<code>-r reserved-blks-count</code>	设置文件系统保留的数据块数量
<code>-T time-last-checked</code>	设置最后一次使用 <code>e2fsck</code> 命令检测文件系统的时间。时间的表示形式采用国际日期格式，即YYYYMMDD[HHMM[SS]]。也可以使用关键字 <code>now</code> ，表示把系统的当前时间作为最后一次检测文件系统的时间
<code>-u user</code>	设置能够使用文件系统保留数据块的用户。指定用户时既可以使用用户ID，也可以使用用户名，但不管采用哪一种指定方式，系统最终都会以数值形式把用户ID写入超级块中

下面是一个例子，说明如何使用`tune2fs`命令，建立目录索引，提高Ext2 文件系统检索大型目录的速度：

```
$ sudo tune2fs -O dir_index /dev/sdb2
tune2fs 1.41.9 (22-Aug-2009)
$
```

如果想在一個Ext2 文件系统中增加日志功能，把Ext2 文件系统转换为Ext3 文件系统，可以使用下列`tune2fs`命令：

```
$ sudo tune2fs -j /dev/sdb2
tune2fs 1.41.9 (22-Aug-2009)
Creating journal inode: done
This filesystem will be automatically checked every 31 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.
$
```

此外，如果想把现有的Ext3 文件系统转换为Ext4 文件系统，可以运行下列`tune2fs`命令（确保文件系统已经卸载）：

```
$ sudo tune2fs -O extents,uninit_bg,dir_index /dev/sdb1
tune2fs 1.41.9 (22-Aug-2009)

Please run e2fsck on the filesystem.

$
```

然后还必须运行下列`fsck`命令，修复文件系统，否则，无法安装刚转换的Ext4 文件系统：

```
$ sudo fsck -Df /dev/sdb1
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
One or more block group descriptor checksums are invalid. Fix<y>? yes

Group descriptor 0 checksum is invalid. FIXED.
Group descriptor 1 checksum is invalid. FIXED.
.....
Group descriptor 7 checksum is invalid. FIXED.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 3A: Optimizing directories
```

```

Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/sdb1: ***** FILE SYSTEM WAS MODIFIED *****
/dev/sdb1: 11/16000 files (0.0% non-contiguous), 7399/63924 blocks
$

```

若想查询Ext2/3/4文件系统的基本信息，可以使用下列tune2fs命令：

```

$ sudo tune2fs -l /dev/sdb1
tune2fs 1.41.9 (22-Aug-2009)
Filesystem volume name:          <none>
Last mounted on:                 <not available>
Filesystem UUID:                 5b808b17-cc4a-4ed9-aae2-62484aea4d97
Filesystem magic number: 0xEF53
.....
Filesystem state:                 clean
Errors behavior:                 Continue
Filesystem OS type:              Linux
Inode count:                     16000
Block count:                     63924
Reserved block count:            3196
Free blocks:                     56542
Free inodes:                     15989
First block:                     1
Block size:                      1024
Fragment size:                  1024
Reserved GDT blocks:             249
Blocks per group:                8192
Fragments per group:            8192
Inodes per group:               2000
Inode blocks per group:          250
.....
First inode:                     11
Inode size:                      128
Journal inode:                   8
.....
$

```

14.4 安装与卸载文件系统

这一节主要介绍怎样在Linux系统中安装和卸载文件系统。

14.4.1 安装文件系统概述

在成功创建了文件系统之后，需要把新建的文件系统安装到Linux文件系统层次目录结构中的某个安装点上，然后才能使用新建的文件系统。对于任何文件系统，包括iso9660格式的CD/DVD、FAT/FAT32类型的DOS文件系统以及NTFS类型的Windows文件系统，都需要先安装，然后才能使用（原始存储介质除外）。

对于CD/DVD、移动盘和U盘等文件系统，Ubuntu Linux系统提供文件系统的自动安装功能。当把上述存储介质插入系统时，Linux系统将会尝试安装相应的文件系统。必要时，也可以在命令行界面中，使用mount和umount命令手动安装与卸载文件系统。

mount命令用于安装文件系统或远程共享资源，umount命令用于卸载文件系统或远程共享资源。除非特别设定，在使用mount命令安装文件系统时，必须具有超级用户的身份，或者使用sudo命令。必要时还需要使用mkdir命令，事先创建适当的安装点。在使用mount/umount命令时通常应注意下列事项：

（1）mount命令不能安装一个已经受损且需要以读写方式安装的文件系统。如果在运行mount命令时出现错误信息，通常需要使用fsck命令检测与修复文件系统。

（2）umount命令不能卸载一个当前正在使用的文件系统，例如：

- 用户正在访问文件系统中的文件或目录，如打开文件，进入某个目录等；
- 文件系统正在共享。

（3）使用remount选项，可以把以只读方式安装的文件系统重新安装为读写方式的文件系统。但不能把读写方式安装的文件系统重新安装为只读方式的文件系统。

（4）对于移动盘一类的文件系统，在用完卸下之前，不要忘记使用umount命令等方式卸载文件系统。

14.4.2 mount命令

安装文件系统的基本命令是mount，其语法格式简写如下：

```
mount [-lhV]
mount { -a | [ [-t fstype] device | directory ] }
mount [-rw] [-t fstype] [-o options] device directory
```

其中，“-a”选项表示安装/etc/fstab文件中列举的所有文件系统。当系统进入多用户等运行模式时，系统将会自动运行“mount -a”命令。“-r”选项表示以只读方式安装指定的文件系统，其作用等同于“-oro”选项。“-w”选项表示以读写方式（默认）安装指定的文件系统，其作用等同于“-orw”选项。“-t”选项用于指定文件系统的类型，如ext2、ext3、ext4、iso9660、vfat及ntfs等。device表示文件系统的设备文件名，如/dev/sda4。directory表示安装点。

上述mount命令可用于不同场合。实际上，由于mount命令的主要功能是把指定设备中的文件系统安装到指定的目录中，故最基本、也是最常用的安装命令的标准语法格式如下：

```
mount -t fstype device directory
```

上述安装命令相当于告诉Linux系统内核，把指定的设备（device）、指定类型（fstype）的文件系统安装到指定的目录（directory）中。

如果省略了“-t”选项，即没有指定文件系统的类型，mount命令需要读取文件系统的超级块或/proc/filesystems文件，最终确定文件系统的类型。在不知道存储设备中的文件系统类型时，可以使用这种方法尝试安装。

安装文件系统之后，在作为安装点的目录中，不管原先存在什么文件和子目录，此时均不可见。安装目录的属主与访问权限也将随之发生变化。从安装文件系统开始，凡是引用安装目录，即相当于引用存储设备中的文件系统的根目录。

在安装文件系统时，针对不同的文件系统类型，可以使用“-o”选项提供文件系统特定的选项。以Ext2/3/4文件系统为例，表14-3给出了使用“mount -o”命令时常用的选项。如果需要同时指定多个选项，可以使用逗号分隔每个选项（注意，选项中间不能有空格），如“-oro, nosuid”。

表14-3 常用的“-o”选项

-o选项	简单描述
async/sync	采用异步或同步方式处理文件系统的所有I/O操作
atime/noatime	允许或禁止更新文件系统中的文件访问时间。通常，每当访问文件时，都会更新信息节点的访问时间字段。noatime选项意味着即使访问文件也不更新信息节点的访问时间属性（此举的主要目的是加速磁盘访问）。在文件修改时间并不重要的情况下，采用noatime选项能够减少文件系统的磁盘处理活动，从而提高磁盘访问的性能。默认值为atime
auto/noauto	auto选项意味着可以使用“-a”选项安装相应的文件系统。noauto意味着只能采用完整的mount命令安装，不能使用“-a”选项安装相应的文件系统
defaults	采用默认的安装选项（rw、suid、dev、exec、auto、nouser和async）安装文件系统
dev/noddev	允许或禁止按文件系统解释字符或块设备
exec/noexec	允许或禁止直接执行正在安装的文件系统中的程序
group	如果用户的用户组ID匹配设备文件的用户组ID，允许普通用户（非超级用户root）安装相应的文件系统
owner	允许设备文件的属主（即使是普通用户）安装与指定设备文件对应的文件系统
remount	重新安装一个已经安装的文件系统。这个选项的主要目的或常见用法是改变已安装文件系统的安装标志，特别是把一个以只读方式安装的文件系统重新安装为可读写文件系统。注意，使用这个选项时既不能更换设备文件，也不能改变安装点
ro/rw	以只读（ro）或读写（rw）方式安装文件系统。默认值为rw（iso9660等文件系统除外）
suid/nosuid	访问远程文件系统时允许或禁止文件的suid或sgid标志位发生作用
user/nouser	允许或禁止普通用户安装与卸载文件系统。安装时，安装用户的名字将会写入/etc/mtab文件，以便同一用户能够卸载文件系统。默认值为nouser，即禁止普通用户安装文件系统
users	允许每个用户安装和卸载文件系统。这个选项蕴含着同时指定了noexec、nosuid和noddev三个选项，除非随后又附加了其他抑制选项，如“users,exec,dev,suid”

14.4.3 fstab文件

在Linux操作系统中，/etc/fstab是一个重要的系统文件，其主要功能是维护系统中安装的文件系统。如果需要在系统引导过程中自动安装某个或某些文件系统，必须把相应的文件系统加到fstab文件中。fstab文件也可用于指定交换（swap）分区，以便在系统引导过程中自动设置系统交换区。

/etc/fstab文件的格式如下，其中每一行包含下列6个字段，表14-4给出了每个字段的简单说明。

```
file-system mount-point type options dump pass
```

表14-4 /etc/fstab文件中的每个字段及简单说明

字段名	简单说明
file-system	表示准备安装的文件系统，其设备文件名可以是下列之一： <ul style="list-style-type: none"> · 文件系统磁盘分区的设备文件名，如/dev/sda7 · 远程文件系统的资源名，其格式为“<host>:<dir>”，如“iscas:/docs” · 以UUID表示的设备文件名 · 用做交换区的磁盘分区，如/dev/sda8 · 虚拟文件系统

字段名	简单描述
mount-point	文件系统的安装点，表示把文件系统安装到哪个目录位置。如果安装点的目录名中包含空格字符，可以使用转义字符“\040”替代
type	标识文件系统的类型。Linux系统支持大量的文件系统，如autofs、ext2、ext3、iso9660、vfat、ntfs、jfs、nfs、sysv、ufs、proc以及tmpfs等。至于当前的Linux系统都支持哪些文件系统，可以查阅/proc/filesystems文件。如果这个字段为swap，表示相应的磁盘分区用做交换区。一个特殊的“ignore”标志表示忽略当前行，意味着暂时停止安装相应的文件系统
options	使用mount命令安装相应的文件系统时需要提供的安装选项，多个安装选项中间需加逗号分隔符（其间没有空格）。有关安装选项的说明详见mount(8)和nfs(5)手册页
dump	dump命令使用这个字段确定相应文件系统中的数据是否需要备份。如果这个字段为0，表示不需要使用dump命令备份相应的文件系统
pass	在启动或重新引导系统时，fsck使用这个字段确定文件系统的检测顺序。“/”文件系统应当总是为1，其他非虚拟文件系统均应为2。位于同一磁盘不同分区中的文件系统只能顺序地执行检测，但对位于不同磁盘中的文件系统，fsck将会采用并发机制，同时进行检测。如果这个字段为0，表示不需要使用fsck命令检测相应的文件系统



注意 /etc/fstab 文件中的每个字段都不能为空，字段值也不允许含有空格；字段之间必须使用空白字符（空格或制表符）作为分隔符。否则会导致安装失败，甚至可能导致系统无法正常启动。

下面是取自Ubuntu Linux系统中的一个fstab 文件实例：

```
$ cat /etc/fstab
.....
# <file system>          <mount point>      <type>          <options>          <dump>  <pass>
proc                      /proc              proc            defaults                    0        0
# / was on /dev/sda2 during installation
UUID=5fd941c4-594..... /                  ext4            relatime,errors=remount-ro 0        1
# swap was on /dev/sda3 during installation
UUID=c8b16352-86f..... none             swap            sw                          0        0
/dev/scd0                 /media/cdrom0      udf,iso9660     user,noauto,exec,utf8      0        0
$
```

14.4.4 安装文件系统

本小节将介绍怎样使用mount命令手动安装文件系统，或通过在/etc/fstab文件中增加安装项的方式自动安装文件系统。Ubuntu Linux系统也能够自动识别与安装CD/DVD以及USB等移动存储设备，只要插入移动存储设备，即可打开一个文件浏览器窗口，显示其根目录中的子目录与文件。但是，软盘或内置硬盘中的文件系统需要手动安装，或在GNOME桌面的“位置”菜单中，通过单击软盘或Windows系统分区菜单项，才能安装相应的文件系统。

在使用mount命令手动安装文件系统时，原则上应利用“-t”选项指定文件系统的类型。如果确实不知道文件系统的类型，或者图省事，也可以不指定，由Linux系统自己确定文件系统的类型。

1. 确定系统中已安装了哪些文件系统

为了确定系统当前已经安装了哪些文件系统，可以使用下列mount命令：

```
$ mount [-l]
```

其中，“-l”选项意味着除了显示基本的安装信息之外，还要显示已安装文件系统的卷标。下面的例子说明了怎样利用mount命令显示系统中当前已安装的文件系统及有关信息：

```
$ mount
/dev/sda2 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw)
none on /sys type sysfs (rw,noexec,nosuid,nodev)
.....
$
```

另外，还可以查阅文件系统安装表/etc/mtab，确定系统中当前已安装了哪些文件系统。无论何时安装或卸载文件系统，Linux系统都会修改文件系统安装表。因此，/etc/mtab文件总是能够反映当前最新的文件系统安装信息。

任何用户均可利用cat或more等命令显示/etc/mtab文件，查询文件系统的安装信息，但任何人都不应直接修改这个文件。mtab文件示例如下：

```
$ cat /etc/mtab
/dev/sda2 / ext4 rw,errors=remount-ro 0 0
proc /proc proc rw 0 0
none /sys sysfs rw,noexec,nosuid,nodev 0 0
.....
$
```

2. 文件系统的自动安装

若想增加一个新的文件系统安装项，使文件系统能够在系统的启动过程中自动安装，需要以超级用户的身份编辑/etc/fstab文件，把相应的文件系统加到/etc/fstab文件中。同时确保每个字段都不为空，必要时可使用mkdir命令，事先为准备安装的文件系统创建一个安装点。

下面的例子说明了怎样修改/etc/fstab文件，以便在系统启动过程中能够利用“mount -a”命令，把磁盘分区/dev/sda4中的文件系统自动安装到/docs目录中。其中的pass字段为2，意味着可以并发检测与修复相应的文件系统。修改后的fstab文件如下，其中最后一行即为新加的文件系统安装项：

```
$ cat /etc/fstab
.....
# <file system>          <mount point>      <type>          <options>          <dump>  <pass>
proc                    /proc              proc            defaults              0        0
# / was on /dev/sda2 during installation
UUID=5fd941c4-594..... /                   ext4            reltime,errors=remount-ro 0        1
# swap was on /dev/sda3 during installation
UUID=c8b16352-86f..... none             swap            sw                    0        0
/dev/scd0               /media/cdrom0      udf,iso9660     user,noauto,exec,utf8 0        0
/dev/sda4               /docs              ext4            rw                    0        2
$
```

3. 使用mount命令安装Ext2/3/4文件系统

如果需要手动安装Ext2/3/4文件系统，可以直接使用下列mount命令：

```
# mount -t fstype device directory
```

其中，fstype是ext2、ext3或ext4。device是文件系统的设备文件名，如/dev/sda4（为了查询文件系统磁盘分区的设备文件名，可以使用fdisk等命令）。directory用于指定安装文件系统的目录位置，也即安装点。

下面的例子说明了怎样利用mount命令，把/dev/sda4中的Ext4文件系统安装到/docs目录中：

```
$ sudo mount -t ext4 /dev/sda4 /docs
$
```

4. 使用mount命令安装FAT文件系统

Ubuntu Linux系统内核支持FAT/FAT32文件系统，其文件系统类型为vfat。如果需要在两个操作系统上交换数据，可以使用软盘、USB移动盘或U盘，把FAT/FAT32文件系统安装到Linux系统中。

Ubuntu Linux系统能够自动识别系统盘中的FAT/FAT32文件系统。在GNOME桌面的“位置”菜单中单击相应的菜单项，即可安装FAT/FAT32文件系统。当插入USB移动盘或U盘时，系统将会自动安装相应的文件系统，同时在GNOME桌面上增加一个图标，打开一个文件浏览器窗口。如果需要手动安装FAT/FAT32文件系统，可以使用下列mount命令：

```
# mount -t vfat device directory
```

其中，device是FAT/FAT32文件系统的设备文件名，如/dev/fd0（软盘）或/dev/sda1等。directory用于指定安装文件系统的目录位置。

Ubuntu Linux系统不会自动安装软盘中的文件系统，GNOME桌面中也没有相应的图标可用（当然也可以使用“位置”菜单中的“软盘驱动器”菜单项）。下面的例子说明了怎样把3.5英寸软盘中的FAT/FAT32文件系统，手动安装到/mnt目录中：

```
$ sudo mount -t vfat /dev/fd0 /mnt
$
```

5. 使用mount命令安装ISO 9660 CD文件系统

在Ubuntu Linux系统中，把CD/DVD插入光驱时，作为iso9660类型的文件系统，系统会自动把CD/DVD安装到/media/cdrom0目录中。如果需要手动安装，可以使用下列mount命令，把CD/DVD安装到指定的目录中：

```
$ sudo mount -t iso9660 -o ro /dev/cdrom /mnt
$
```

6. 使用mount命令安装NTFS文件系统

Ubuntu Linux系统内核直接支持NTFS文件系统，能够自动识别位于系统盘中的NTFS文件系统。在GNOME桌面的“位置”菜单中单击相应的菜单项，即可以NTFS文件系统的卷标（如“Local Disk”）作为子目录名，安装到“/media/local Disk”目录中。

采用手动安装方式时，可以把NTFS文件系统安装到任何目录位置，如常用的/mnt目录等。如果同时访问的NTFS分区较多，可能需要创建若干安装点，如/media/c、/media/d、……等，以

便能够把每个NTFS文件系统分别安装到Linux文件系统的不同目录层次结构中。例如, 采用下列命令可以把Windows系统的“C.”盘安装到/mnt目录中:

```
$ sudo mount -t ntfs /dev/sda1 /mnt
$
```

在安装之前, 为了查询系统盘中NTFS文件系统的设备名, 可以使用下列fdisk命令, 列出系统中的所有NTFS磁盘分区:

```
$ sudo fdisk -l /dev/sda | grep NTFS
/dev/sda1 * 1 1824 14651248+ 7 HPFS/NTFS
$
```

7. 使用mount命令安装NFS文件系统

假定iscas是一个NFS服务器, 为了把其中提供的共享目录/share/tools, 作为NFS文件系统安装到本地系统的/tools目录中, 可以使用下列命令:

```
$ sudo mount -t nfs -o ro iscas:/share/tools /tools
$
```

14.4.5 卸载文件系统

卸载文件系统意味把文件系统从安装点移走, 删除/etc/mtab文件中的安装项。在文件系统的管理与维护任务中, 部分处理任务只能在未安装的文件系统中执行。此外, 当临时安装的文件系统不再需要时, 也应及时卸载文件系统。因此, 当出现下列情况时, 应当考虑卸载文件系统:

- 文件系统用过之后已不再需要;
- 文件系统受损, 因而需要使用fsck命令检测与修复文件系统;
- 为了增加卷标、目录索引以及日志功能, 因而需要使用tune2fs命令调整文件系统;
- 删错文件, 因而需要使用debugfs命令调试文件系统;
- 在执行完整的文件系统备份之前, 通常也需要卸载文件系统。



注意 作为系统关机过程的一部分, 文件系统会自动卸载。

卸载文件系统时, 可以使用umount命令。umount命令的语法格式简写如下:

```
umount [-flnrv] device | directory
umount -a [-flnrv] [-t fstype]
```

其中, “-f”选项表示强制卸载指定的文件系统; “-a”选项表示卸载/etc/mtab文件中列举的所有文件系统(/proc文件系统除外); “-t”选项表示卸载指定类型的文件系统; device是文件系统的设备文件名或NFS远程文件系统的资源名等; directory是准备卸载的文件系统安装点。

1. 卸载文件系统前的准备工作

在卸载文件系统之前, 通常需要做一定的准备工作, 或具备一定的条件, 其中包括:

- (1) 通常情况下, 只有超级用户才能卸载文件系统。

（2）文件系统必须处于空闲状态才能够卸载。通常，不能卸载尚处于工作状态的任何文件系统。所谓文件系统处于工作状态，意味着可能还有用户正处于文件系统中的某个目录，正在访问其中的文件，或者文件系统正处于共享状态。为了确保文件系统能够正常卸载，可以采取下列措施：

- 改换到不同文件系统上的目录；
- 退出Linux系统；
- 使用fuser命令找出访问文件系统的所有进程，然后通知用户停止使用准备卸载的文件系统，或由超级用户终止相应的进程（如果必要或可能）；
- 取消文件系统的共享。

2. 终止正在访问文件系统的所有进程

如果想要终止正在访问文件系统的所有进程，首先应使用下列命令，显示当前正在访问文件系统的所有进程，以便了解需要终止运行哪些进程：

```
# fuser -c [ -u ] mount-point
```

其中，“-c”选项表示显示当前正在访问指定安装点（文件系统）中任何文件的所有进程。“-u”选项意味着在显示的每个进程ID后面给出与进程相关的注册用户名。“mount-point”表示需要检查的安装点（文件系统）。

然后，可以使用下列命令，终止运行当前正在访问文件系统的所有进程：

```
# fuser -c -k mount-point
```

在执行上述命令时，一个SICKILL信号将会发送到当前正在访问指定文件系统的每个进程。注意，在事先没有通知用户时，通常不应强行终止用户的进程。

最后，可以使用下列命令，验证当前是否还有任何进程正在访问文件系统：

```
# fuser -c mount-point
```

下面的例子说明了怎样停止一个正在访问/docs文件系统的进程6968：

```
$ sudo fuser -c -u /docs
/docs: 6968c(gqxing)
$ sudo fuser -c -k /docs
/docs: 6968c
$ sudo fuser -c /docs
$
```

3. 卸载文件系统

除了“/”文件系统（包括单独的/usr等文件系统）之外，如果想要卸载其他非虚拟文件系统，首先应确保已完成文件系统卸载前的准备工作，然后可以使用umount命令卸载文件系统。注意，仅当关闭系统时才能卸载“/”文件系统，否则系统无法正常运行。

下面的例子说明了怎样卸载一个安装在/mnt目录的本地文件系统：

```
$ sudo umount /mnt
$
```

下面的例子说明了怎样卸载一个位于磁盘分区4中的文件系统：

```
$ sudo umount /dev/sda4
$
```



注意

上述命令并不能卸载当前正处于工作状态的文件系统。在紧急情况下，可以使用“`umount -f`”命令强制卸载一个尚在工作的文件系统。除非不得已，通常不建议采取这种卸载方式，因为这种强制卸载有可能会造成已打开的文件丢失数据。下面的例子说明了怎样强制卸载位于安装点/`mnt`处的文件系统：

```
$ sudo umount -f /mnt
$
```

14.5 检测与修复文件系统

`fsck`或`e2fsck`命令可用于交互检测与修复文件系统。在实施修复之前，`fsck`命令通常会提示用户确认建议的处理动作。

下面以Ext2/3/4文件系统为例，说明怎样利用`fsck`或`e2fsck`命令实现文件系统的检测与修复。`fsck`或`e2fsck`命令的主要功能是检测文件系统的完整性，对发现的问题尝试进行适当的修复。`fsck`和`e2fsck`命令的语法格式简写如下：

```
fsck [-ANV] [-Dfnprvy] [-t fstype] [-b superblock] [filesystem]
e2fsck [-DfnprvVy] [-b superblock] [-B blocksize] filesystem
```

其中，`filesystem`可以是一个磁盘分区的设备文件名，如`/dev/sdb2`，可以是一个文件系统的安装点，如`/var`，也可以是一个Ext2/3/4文件系统的卷标或UUID。如果命令行中未指定文件系统，也未指定“-A”选项，`fsck`将依次检测`/etc/fstab`文件中列举的所有文件系统。`fsck`与`e2fsck`命令的其他常用选项如表14-5所示。

表14-5 `fsck`与`e2fsck`命令的常用选项

选项	简单描述
-A	遍历 <code>/etc/fstab</code> 文件，同时检测其中列举的所有文件系统。除非指定了“-P”选项，首先检测“/”文件系统。然后按照 <code>/etc/fstab</code> 文件 <code>pass</code> 字段指定的顺序，依次检测每个文件系统。如果 <code>pass</code> 字段为0，则跳过相应的文件系统。 <code>pass</code> 字段的值确定了文件系统检测的先后顺序，数值越小，检测的顺序越靠前。如果多个文件系统的 <code>pass</code> 字段具有相同的值， <code>fsck</code> 将会尝试并行检测相应的文件系统。在 <code>/etc/fstab</code> 文件中，通常把“/”文件系统的 <code>pass</code> 字段设置为1，其他文件系统均设置为2，以便能够充分利用 <code>fsck</code> 的并行自动检测功能
-b <i>superblock</i>	使用指定的备份超级块（而不是常规的主超级块）检测与修复文件系统。当文件系统的主超级块严重受损时，可以使用这个选项修复文件系统。备份超级块的位置依赖于具体的文件系统，以及文件系统的大小与逻辑数据块的大小等。在逻辑数据块为1 KB的文件系统中，第一个备份超级块位于8193号数据块；在逻辑数据块为2 KB的文件系统中，第一个备份超级块位于16384号数据块；在逻辑数据块为4 KB的文件系统中，第一个备份超级块位于32768号数据块。为了确定其他备份超级块的位置，可以使用“ <code>mke2fs -n</code> ”命令查询其输出结果
-B <i>blocksize</i>	通常， <code>e2fsck</code> 将会尝试以各种不同的数据块规格检索和确定超级块的位置与大小。“-B”选项强制 <code>e2fsck</code> 按照指定的超级块规格找出超级块。如果指定规格的超级块不存在， <code>e2fsck</code> 将会输出一个错误信息，然后结束程序的执行

(续表)

选项	简单描述
-D	优化文件系统中的目录。利用这个选项，通过建立索引或重建索引（如果文件系统支持目录索引，参见tune2fs命令），排序或压缩目录，或采用传统的线性目录，e2fsck将会尝试优化所有的目录
-f	强制执行文件系统的检测，即使文件系统完好无损
-n	以只读方式打开指定的文件系统，对于fsck的所有修复请求，均以“no”作为响应，使fsck能够以非交互的方式自动检测指定的文件系统，报告（而不是修复）文件系统中存在的问题
-N	实际上并不执行，只是向用户展示，一旦执行时fsck命令究竟能够做什么
-p	并发地检测文件系统，自动地修复部分简单的问题，而无需用户干预。一旦检测到严重的错误，立即结束文件系统的检测与修复。如果发现某个问题需要提请用户做进一步的校正处理，fsck或e2fsck将会输出一个错误描述信息，然后结束程序的执行
-r	交互地检测与修复文件系统，在修复之前请求用户予以确认。注意，这也是e2fsck默认的文件系统检测与修复方式
-t <i>fstype</i>	指定文件系统的类型
-y	对于fsck的所有修复请求，均以“yes”作为响应，使fsck或e2fsck能够以非交互的方式自动检测与修复指定的文件系统

14.5.1 何时需要检测文件系统

1. 文件系统受损的外部原因

Ext2/3/4等文件系统主要依赖于超级块，用以跟踪和管理信息节点与数据块的分配与释放。当系统内存中的超级块与磁盘文件系统上的超级块没有适当地同步时，就会导致磁盘上的超级块与文件系统实际数据不一致。此时的文件系统就需要修复，否则无法正常安装文件系统。

由于下列原因而未正常停止操作系统时，将会造成超级块与实际数据不一致，从而破坏文件系统：

- 电源故障，包括突然断电、人为造成的电源故障以及内部电源故障等；
- 强行关机，即在未正常停止Linux系统之前就断开电源；
- 硬件故障，如磁盘出现坏块等；
- 因Linux系统内核故障而引起的异常关机。

一旦文件系统受损，按照/etc/fstab文件的设置，在引导过程中，系统都会自动执行fsck命令，检测文件系统的完整性。在文件系统的检测过程中，fsck将会尝试修复已经受损的文件系统，把修复后的文件系统安装到指定的目录位置。如果文件系统损坏严重，fsck可能无法完全修复，从而可能导致系统无法正常启动。

在文件系统的修复过程中，fsck命令将会把已分配但未引用的文件与目录置于lost+found目录下，且以其信息节点号作为文件或目录的名字。如果lost+found目录不存在，fsck命令将会自动创建一个。如果lost+found目录中没有足够的空间，fsck命令将会尝试增加其容量。

2. Ext2/3/4文件系统受损的内部因素

在一个正常运行的Linux系统中，每个工作日都可能会创建、修改或删除数百个文件。每当修改文件时，操作系统都会执行一系列文件系统更新操作。这些更新如果能够及时地全部写

到磁盘上，就不会破坏文件系统的完整性。通常，磁盘数据的更新是异步处理的，当用户进程访问文件系统，如写文件数据时，这些数据首先被复制到内存缓冲区中。此时，系统允许用户进程继续写数据，即使先前的数据尚未真正写到磁盘中。

因此，在任何给定时刻，磁盘文件系统的状态总是滞后于内存超级块中表示的文件系统。当缓冲区需要分配他用，或按一定的时间间隔，系统把缓冲区中的数据写到磁盘文件系统时，磁盘上的文件系统才能真正等同于内存超级块表示的文件系统。如果在停机之前没有把内存超级块中的数据写到磁盘上，磁盘上的文件系统就会出现不一致的状态，造成文件系统受损。

3. 文件系统状态标志字段

超级块中有一个记录文件系统状态的标志字段，fsck命令使用这个标志字段确定文件系统的状态，决定是否需要检测文件系统的完整性。如果这个标志字段状态完好，即使文件系统的超级块存在其他问题，fsck等命令也不会检测文件系统。

14.5.2 文件系统检测的内容

在检测Ext2/3/4文件系统时，fsck命令将会针对文件系统的超级块、数据块组描述、信息节点位图、数据块位图、信息节点、间接地址块以及数据块等重要组成部分，检测其完整性与一致性。注意，fsck并不验证文件本身的数据内容。

1. 超级块

超级块（包括数据块组描述、信息节点位图与数据块位图）是文件系统的核心，其中存有文件和目录的关键数据与汇总信息。每当文件数据发生变动时都需要更新超级块、数据块组描述、信息节点位图与数据块位图。因此，超级块、数据块组描述、信息节点位图与数据块位图是文件系统中更新最频繁的部分，通常也最容易受到损坏。

所谓文件系统受损，主要是指超级块（包括数据块组描述等）中的数据受损。每当改动文件系统的信息节点或数据块时，都需要修改超级块、数据块组描述、信息节点位图和数据块位图。如果卸载文件系统之前执行的最后一个文件系统操作不是sync命令，几乎肯定会损坏超级块。

超级块的完整性检测主要包括以下四个方面：

- 文件系统的大小；
- 信息节点的数量；
- 空闲数据块计数；
- 空闲信息节点计数。

检测文件系统和信息节点表的大小

文件系统的大小和信息节点区的大小均属于静态数据，一旦创建文件系统之后，这些数据是不会改变的。因此，fsck命令实际上没有办法严格检测这种数据，因为这些数据在创建文件系统时就已确定。但是，fsck命令能够依据合理的推断，检测这些静态数据的有效性。原则上，文件系统的大小必须大于超级块与信息节点表示的数据块数，信息节点的数量必须小于文件系统允许的最大值。信息节点含有除文件名与信息节点号之外的所有文件属性信息，文件系统的大小和信息节点数量等统计数据是fsck命令最为关注的重要信息。针对文件系统的所有检测都要求这些数字必须是正确的。如果检测到主超级块的静态数据有误，fsck命令将会要求操作员指定备份超级块的位置。

空闲数据存储块检测

空闲数据块是在数据块组的数据块位图中维护的，fsck命令将会检测所有未分配的空闲存储块。在计数了所有空闲存储块之后，fsck命令将会据此确定空闲存储块的数量加上信息节点已声明占用的数据块数量是否等于文件系统全部存储块的总数。如果发现任何数据块组中的数据块位图有问题，fsck命令将会适当地修正数据块位图，剔除分配有误的空闲存储块。

超级块的汇总信息含有文件系统中所有空闲存储块的统计数据，fsck命令将会根据此计数，与文件系统检测过程中发现的实际空闲存储块总数进行比较。如果两个数字不一致，fsck命令将会以实际的空闲存储块计数替换超级块中的统计数字。

空闲信息节点检测

超级块中的汇总信息也含有文件系统中所有空闲信息节点的统计数据，fsck命令也会根据此计数，与文件系统实际空闲信息节点总数进行比较。如果两个数字不一致，fsck命令将会用实际的空闲信息节点数量替换超级块中的统计数字。

2. 信息节点

对信息节点的完整性检测将从信息节点表的2号信息节点（1号保留）位置开始，顺序检测每个信息节点的完整性。信息节点的完整性检测包括以下几方面的内容：

- 类型与状态；
- 链接计数；
- 重复数据块；
- 无效数据块；
- 信息节点中的数据块计数；
- 信息节点的连接性。

类型与状态检测

信息节点中的mode字段描述了文件的类型与信息节点的状态。每个信息节点可处于下列三种状态之一：

- 信息节点已经分配或占用（其中的mode字段为非0）；
- 信息节点尚未分配或空闲（其中的mode字段为0）；
- 信息节点部分分配（信息节点中必须具备的字段不完整）。

创建文件系统时，Linux系统将会预留一定数量的信息节点，而且仅当必要时才会分配这些信息节点。已经分配的信息节点会指向相应的文件，未分配的信息节点没有指向任何文件，因此其中（如地址字段）应是空的，而处于部分分配状态的信息节点意味着其中的信息是不正确的。出现这种状态的一种可能是由于硬件故障致使垃圾数据写入信息节点。针对此类情况，fsck命令应当采取的唯一正确动作就是清除这样的信息节点。

链接计数检测

每个信息节点均包含一个链接到当前信息节点的文件或目录项计数。fsck命令将会从根目录开始考察整个目录结构，计算每个信息节点的实际链接计数，以验证每个信息节点的链接计数是否正确。信息节点中的链接计数与fsck命令确定的实际链接计数之间的差异可以归结为下列三种情况之一：

- 信息节点中的链接计数不为0，但实际计数为0。如果信息节点对应的文件或目录项不存在，就会出现这种差异。在此情况下，fsck命令将会把没有归属的（断开链接的）文件置于lost+found目录中。
- 信息节点中的链接计数不为0，实际计数也不为0，但两个计数并不相同。如果已经增加或删除了某个目录项，但信息节点尚未更新，就会出现此类差异。在这种情况下，fsck命令将会使用实际的链接计数替换信息节点中的链接计数。
- 信息节点中的链接计数为0，但实际的链接计数不为0。在此情况下，fsck命令将会把信息节点中的链接计数改为实际的链接计数。

重复数据块检测

在文件系统中，每个数据块只能属于一个信息节点，或为空闲数据块，两者只能居其一。如果信息节点中的一个数据块同时属于另外一个信息节点或空闲数据块区，可认为此数据块是重复的。

每个信息节点中的地址数组均直接或间接指向归属于信息节点的所有数据块。因为间接地址块也归属于信息节点，如果间接地址块的归属也出现问题，将会直接影响拥有间接地址块的信息节点。

fsck命令将会对归属于信息节点的每个数据块号与已分配数据块位图进行比较。如果其他信息节点也声明拥有其中的某个数据块号，fsck将会把该数据块号置于重复数据块表中。否则，更新已分配数据块位图，使之包括相应的数据块号。

如果发现重复数据块，fsck命令将会对信息节点位图进行第二次扫描，以便找出声明拥有每个重复数据块的其他信息节点。在此情况下，fsck命令本身并无法肯定究竟哪一个信息节点有误，因此fsck命令将会提示用户选择保留哪一个信息节点，清除哪一个信息节点。注意，如果信息节点中出现大量的重复数据块，通常都是由于间接地址块没有正确写入文件系统引起的。

无效数据块检测

在文件系统中，所有数据块都是从0开始顺序编号的。超级块中的文件系统大小确定了最大数据块编号。编号超出这个范围的数据块均可看做无效数据块，或简称坏块。

fsck命令将会检测信息节点声明拥有的每一个数据块号，确定其编号是否大于文件系统中第一个数据块号且小于等于最后一个数据块号。如果数据块号超出了此范围，可认为这是一个无效数据块。

导致信息节点中出现无效数据块的原因有可能是间接地址块没有写入文件系统。对此，fsck命令将会提示用户清除这样的信息节点。

信息节点中的数据块计数检测

每个信息节点均包含一个数据块引用计数。实际的数据块计数应是已分配的数据块与间接地址块之和。fsck命令将会据此计算数据块的数量，然后与信息节点声明拥有的数据块数进行比较。如果信息节点中的数据块计数有误，fsck命令将会提示用户予以修正。

每个信息节点也包含一个64位的表示文件大小的字段，这个字段给出了相应文件中的字节数。利用这个字段的字节数可以概略计算出相应文件应当占用多少个数据块，然后再用这个数值与信息节点声明拥有的实际数据块数进行比较，即可得出信息节点中的数据块计数是否有误。

信息节点的连接性检测

每个已分配的信息节点都应当存在于文件系统中的某个目录中。如果某个信息节点没有任何引用关系，即任何目录中都没有引用这个信息节点，也非空闲信息节点，可认为此信息节点已经游离于文件系统，因而称做“未引用的”信息节点。

3. 目录结构

目录与普通文件的区别主要依据信息节点中的mode字段。目录数据块含有一系列目录项。对目录数据块的完整性检测主要包括以下三个方面：

- 未分配信息节点检测，即检测目录项中的信息节点号是否指向一个尚未分配的信息节点；
- 无效信息节点检测，即检测目录项中的信息节点号是否大于文件系统中信息节点的数量；
- 不正确的“.”和“..”目录项检测，即检测“.”和“..”目录是否存在不正确的信息节点号；
- 游离目录检测，即检测是否存在游离的目录。

尚未分配的信息节点检测

如果目录数据块中的信息节点号指向一个尚未分配的信息节点，fsck命令将会删除相应的文件或目录项。如果包含新目录项的目录数据块已经修改并写到磁盘文件系统中，但相应的信息节点却没有写到磁盘文件系统中，就会出现此类问题，这种情况通常是由于意外关机造成的。

无效信息节点检测

如果目录项中的信息节点号超出了信息节点区的范围，fsck命令将会删除相应的目录项。当由于某种原因把垃圾数据写入目录数据块时，就会出现这种情况。

不正确的“.”和“..”目录项检测

“.”目录项的信息节点号必须位于目录数据块中的第一个目录项，且该目录的信息节点号必须指向自身。也就是说，其数值必须等于目录的信息节点号。

“..”目录项的信息节点号必须位于目录数据块中的第二个目录项，且此信息节点号必须等于父目录或自身（如果是根目录）的信息节点号。

如果“.”和“..”目录的信息节点号有误，fsck命令将会使用正确的数值予以替换。如果目录存在多个硬链接，fsck将会把第一个发现的硬链接看做“..”目录应当指向的真正父目录。在此情况下，fsck命令会建议用户删除其他名字。

游离目录检测

fsck命令还将检测整个文件系统的连接关系。如果发现某个目录与文件系统没有任何连接关系，fsck命令将会把相应的目录置于文件系统的lost+found目录中。当由于某种原因仅仅把信息节点写入文件系统，但尚未把相应的目录数据块及时写入文件系统时，就会出现此情况。

4. 间接地址块

间接地址块也归信息节点拥有。因此，间接地址块的问题将影响相应的信息节点。fsck检测的间接地址块问题主要包括以下两个方面：

- 多个信息节点声明拥有同一间接地址块；
- 间接地址块的块号超出文件系统的范围。

5. 数据块

信息节点可以直接或间接引用三种数据块。

- 普通数据块;
- 符号链接数据块;
- 目录数据块。

其中, 从属于文件的普通数据块含有文件的实际数据内容, fsck命令不会检测普通文件数据块中数据内容的有效性。符号链接数据块含有符号链接文件中存储的实际路径名。目录数据块含有目录中的所有目录项, fsck命令只能检测目录数据块的有效性。

14.5.3 交互检测与修复文件系统

遇到下列情况时, 通常需要交互检测与修复文件系统:

- 文件系统无法安装;
- 文件系统在使用过程中报错。当文件系统在运行过程中报错时, 错误信息可能会出现在控制台上, 或者写到系统的错误信息日志文件中(严重时甚至可能会引起系统瘫痪)。

在运行fsck命令检测和修复文件系统之前, 应当记住下列注意事项:

- 在使用fsck命令检测文件系统期间, 相应的文件系统应处于非工作状态。否则, 当内存超级块定时更新磁盘超级块时, 会导致文件系统不断发生变化, 致使fsck无法准确地检测文件系统。尤其严重的是, 超级块的更新与fsck的修复将会相互影响, 造成超级块数据的混乱, 因此极有可能损害甚或进一步破坏文件系统, 严重时甚至会危及系统的正常运行。
- 在使用fsck命令检测文件系统之前, 应卸载相应的文件系统。确保文件系统的数据结构处于完好状态。唯一的例外是“/”文件系统(包括单独的/usr文件系统), 因为这个文件系统必须总是安装且处于工作状态, 否则无法运行fsck命令(但可考虑使用其他方式引导系统, 参见下面的说明)。
- 如果需要修复“/”文件系统, 应尽可能使用其他设备或方式引导系统, 以便这些文件系统能够处于非安装或非工作状态。
- 此外, 只有超级用户才能使用fsck命令检测与修复文件系统。因此, 在检测与修复文件系统时, 必须拥有超级用户的访问权限。

检测和修复其他文件系统的步骤如下:

- (1) 卸载文件系统, 然后使用相应的设备文件名作为fsck命令的参数, 检测文件系统。
- (2) 根据fsck命令的提示, 修正fsck命令报告的错误, 校正与修复文件系统。
- (3) 如果执行一次fsck命令并未完全修复所有的错误, 可再次运行fsck命令, 重复检测文件系统。但如果多次运行fsck命令仍然不能修复所有的问题, 可参考14.5.6小节的说明。
- (4) 安装已修复的文件系统, 检查lost+found目录中是否存在任何文件(fsck命令置入lost+found目录中的文件均按其信息节点号命名)。
- (5) 如果存在fsck命令收集的文件, 可以使用file等命令确定文件的类型, 再利用grep、cat或od等命令检查其内容, 在重新命名能够挽回的文件之后, 把这些文件移至适当的目录。最后, 删除或备份lost+found目录中剩下的暂时无法处理的文件或目录, 以免占用lost+found目录的存储空间。

如果是“/”文件系统有问题，应尽可能采用重新引导系统的方式，由Linux系统自动修复文件系统。如果重新启动系统仍然无法修复文件系统，可在引导系统时选择“Ubuntu version, kernel release-generic(recovery mode)”，或使用CD/DVD等安装介质把系统引导至系统维护模式，然后按照上述步骤进行恢复。

下面的例子说明了怎样检测与修复一个文件系统，校正其目录大小有误的问题（在响应“Fix<y>?”询问时，可输入“y”，接着按Enter键，或直接按Enter键）：

```
$ sudo fsck -t ext4 /dev/sdb1
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
/dev/sdb1 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Inode 15, i_size is 512, should be 1024. Fix<y>? yes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

/dev/sdb1: ***** FILE SYSTEM WAS MODIFIED *****
/dev/sdb1: 15/16000 files (6.7% non-contiguous), 7405/63924 blocks
$
```

14.5.4 自动检测与修复文件系统

利用“fsck -p”命令，可以任由fsck命令自动检测与修复由于非正常关机而引起的文件系统受损问题。注意，如果遇到需要由操作员干预的问题，fsck命令将会输出一个简单的错误描述，然后立即停止运行。此外，“fsck -p”命令能够并发地检测文件系统。

下面的例子说明了怎样自动检测与修复一个文件系统：

```
$ sudo fsck -t ext4 -p /dev/sdb1
fsck from util-linux-ng 2.16
/dev/sdb1 was not cleanly unmounted, check forced.
/dev/sdb1: Inode 2, i_blocks is 6, should be 2. FIXED.
/dev/sdb1: Inode 12 ref count is 2, should be 1. FIXED.
/dev/sdb1: 15/16000 files (0.0% non-contiguous), 7394/63924 blocks
$
```

采用交互方式检测文件系统时，也可以使用“-y”选项实现文件系统的自动检测。fsck命令的“-y”选项表示，对于fsck命令提出的每一项修复建议，总是以“yes”给予确认。例如：

```
$ sudo fsck -t ext4 -y /dev/sdb1
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
/dev/sdb1 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
/lost+found not found. Create? yes

Pass 4: Checking reference counts
Pass 5: Checking group summary information
```



```
Block bitmap differences: -266
Fix? yes

/dev/sdb1: ***** FILE SYSTEM WAS MODIFIED *****
/dev/sdb1: 15/16000 files (0.0% non-contiguous), 7394/63924 blocks
$
```

14.5.5 恢复严重受损的超级块

当文件系统的超级块已严重受损，文件系统无法正常安装时，必须采用备份超级块才能恢复文件系统。Ext2/3/4文件系统存在多个超级块副本，可以利用fsck命令的“-b”选项，选用任何一个超级块副本替换主超级块。

首先使用下列mkfs命令显示超级块中的数据（注意一定要加“-n”选项，如果忽略了这个选项，将会毁灭文件系统中的所有数据，而代之以一个空的文件系统）：

```
# mkfs -t fstype -n device-name
```

上述命令将会显示之前使用mkfs命令创建文件系统时用做超级块副本的数据块号。

然后使用下列fsck命令，同时给出超级块副本，使之替换受损的超级块：

```
# fsck -t fstype -b block-number device-name
```

上述fsck命令将会使用指定的超级块副本恢复主超级块。通常情况下，不管文件系统的规模如何，几乎总是能够使用8193号数据块作为备份超级块，参见“mkfs -t fstype -n device-name”命令的输出信息。

如果“/”等文件系统上的超级块严重受损，导致系统无法正常启动，此时至少有两种选择：在引导系统时选择“Ubuntu version, kernel release-generic(recovery mode)”，或利用CD/DVD安装介质引导系统进入维护模式，然后再按照上述步骤进行恢复。严重时可能需要重装Linux系统。

下面的例子说明了怎样使用mkfs命令获取超级块副本，然后利用fsck命令和第一个超级块副本，即8193号数据块，修复超级块受损的文件系统：

```
$ sudo mkfs -t ext4 -n /dev/sdb1
mke2fs 1.41.9 (22-Aug-2009)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
16000 inodes, 63924 blocks
3196 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=65536000
8 block groups
8192 blocks per group, 8192 fragments per group
2000 inodes per group
Superblock backups stored on blocks:
    8193, 24577, 40961, 57345

$ sudo fsck -t ext4 -b 8193 /dev/sdb1
fsck from util-linux-ng 2.16
```

```
e2fsck 1.41.9 (22-Aug-2009)
data01 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
Free blocks count wrong for group #0 (7675, counted=7672).
Fix<y>? yes

Free blocks count wrong (60638, counted=60635).
Fix<y>? yes

data01: ***** FILE SYSTEM WAS MODIFIED *****
data01: 14/16000 files (0.0% non-contiguous), 3289/63924 blocks
$
```

14.5.6 其他文件系统修复方法

fsck命令采用多次扫描分析的处理方式，而后期扫描分析时修正的错误可能还会暴露出需要在早期扫描分析阶段才能检测和处理的其他问题。因此，有时需要多次运行fsck命令，直至fsck命令不再报错。这种做法能够确保发现和修复所有的错误。但是，fsck命令不会自己连续地运行直至排除所有的错误，因此必须手动重复输入命令。

用户应当关注fsck命令显示的各种信息，这些信息有助于修正错误。例如，假定有一条信息指出某个目录已经损坏，如果删除该目录，fsck命令修复文件系统的工作可能很快就会成功地结束。

如果文件系统无法完全修复，但能够以只读方式进行安装，可以尝试利用cp、tar或cpio等命令从文件系统中取出部分或所有数据。

如果fsck命令无法完全修复文件系统，可以尝试利用debugfs等命令修正错误。在最坏的情况下，也许需要采用mkfs命令重建文件系统，然后再利用备份介质恢复数据。

如果由于磁盘的硬件故障导致文件系统错误，也许需要重新格式化磁盘（分区）或重建文件系统，利用mkfs命令的“-c”选项，剔除坏块，然后再恢复其中的用户数据。注意，如果是硬件错误，通常会重现甚至多次重复显示相同的错误信息。

14.5.7 fsck的处理方式

通常，当系统非正常停机，导致文件系统的最新变化没有及时更新磁盘时，在重新引导系统的过程中，fsck命令将会以非交互的方式，自动检测并修复文件系统。但这种修复只能解决基本的问题，一旦遇到严重的情况，fsck不会进一步尝试自行修复，而是报告出错信息，然后停止运行。

当用户以交互方式运行fsck命令时，fsck将会报告检测到的所有错误，并自行解决自己能够修复的简单问题。对于比较严重的错误，fsck命令将会给出简明的错误信息，同时提请用户做出选择。当使用“-y”或“-n”选项运行fsck命令时，用户的选择总是按照预定的“yes”或“no”响应fsck命令的每一个提问。

在这种文件系统的修复过程中，某些校正动作可能导致数据的丢失。数据丢失的数量和严重程度可以从fsck命令的输出信息中做出判断。

在开始正式检测文件系统之前，fsck 首先会执行命令行语法检查，然后申请内存空间，设置各种工作表。如果存在，此时的错误信息主要涉及用户提供的命令行选项是否正确，以及运行fsck命令时的内存分配申请是否能够得到满足等。上述初始化阶段的任何错误都会终止以非交互形式运行的fsck命令。

fsck命令是一个采用多阶段分析处理方式检测文件系统的程序。在完成初始化之后，fsck命令将分5个阶段，分别检测信息节点、数据块与文件的大小，检测文件的路径名、连接性、引用计数，以及信息节点与数据块位图，做出必要的纠正，或请求用户做出选择。各阶段检测处理的具体内容如下：

- 第一阶段，检测信息节点、数据块与文件的大小；
- 第二阶段，检测目录的结构及目录项；
- 第三阶段，检测目录的连接性；
- 第四阶段，检测信息节点的链接计数；
- 第五阶段，检测数据块组及文件系统的汇总信息。

下面是以交互方式运行fsck命令，在成功结束文件系统检测之后给出的信息：

```
$ sudo fsck -t ext4 /dev/sdb1
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
data01 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Inode 12 ref count is 2, should be 1. Fix<y>? yes
Pass 5: Checking group summary information

data01: ***** FILE SYSTEM WAS MODIFIED *****
data01: 14/16000 files (0.0% non-contiguous), 3289/63924 blocks
$
```

检测与修复文件系统的处理过程需要大量访问磁盘，Ext2/3/4文件系统采用了一种优化的算法，能够避免重复访问文件系统的结构数据，且在检测处理的过程中，fsck将会按照数据块地址号对信息节点和目录进行排序，以便随后的处理能够减少磁头的移动，提高磁盘访问的效率。

1. 第一阶段：检测信息节点、数据块计数与文件的大小

在第一阶段，fsck会遍历文件系统中的所有信息节点，检测每一个信息节点的连接状态，除信息节点之外，这种检测不会对其他任何对象进行交叉检测。例如，此阶段的检测只是确保文件的模式字段是合法的，信息节点中涉及的所有数据块号地址都是合法有效的。最后，fsck本身也采用数据块位图和信息节点位图的方式，把所有已分配的数据块和信息节点汇集到一起，以便为后续阶段的其他检测做好准备。

如果fsck发现某个数据块存在多个信息节点同时引用的情况，将会调用“Pass 1B”直至“Pass 1D”等多个辅助处理过程解决此类冲突：或者复制共享的数据块，以便每个信息节点都有一个共享数据块的副本；或者从信息节点中删除重复的引用。

第一处理阶段的处理过程需要较长的时间，因为fsck需要把所有信息节点都读入内存并进

行检测。为了减少之后的I/O时间，fsck将会把所有重要的文件系统信息（如文件系统内每个目录数据块的磁盘位置等）缓存在内存中，以避免在第二个阶段（Pass 2）处理目录时再重新读取目录的信息节点结构及其相关数据。

这个阶段主要检测信息节点表，考察信息节点的组成部分（如文件类型、数据块计数、文件的大小以及信息节点结构的格式）是否有误，检测并修复下列问题：

- 检测信息节点相应文件的类型；
- 考察信息节点指向的数据块中是否存在无效数据块，不同信息节点是否包含重复数据块，分析并处理重复的数据块；
- 检测信息节点相应文件或目录的大小；
- 检测信息节点的格式。

2. 第二阶段：检测目录的结构及目录项

第二阶段主要检测目录的结构数据。任何一个目录项只能存在于目录文件的某个数据块中，不可能涉及多个数据块，因此，可以单独检测每个目录文件的数据块，使fsck能够按块号地址对目录文件的所有数据块进行排序，并按升序检测每个目录文件的数据块，从而减少磁头的移动，提高检测的效率。检测目录的目的是确保所有目录项都是合法有效的，其引用的信息节点号也确实是当前正在使用的信息节点。

在检测目录时，针对每个目录信息节点的第一个目录数据块，首先验证其中的当前目录“.”与父目录“..”两个目录项是否确实存在，确认当前目录项“.”中的信息节点号与当前实际目录的信息节点号是否匹配（此时暂不考虑父目录项“..”，而是留在第三阶段处理）。

在第二阶段的检测过程中，fsck将会把每个目录的父目录信息缓存在内存中。如果目录间存在多个引用关系，fsck将会把多余的引用做为不合法的硬链接予以删除。

值得一提的是，在第二阶段的后期处理过程中，fsck需要执行的所有磁盘I/O操作几乎都已完成。第三、第四和第五阶段检测需要的信息均缓存在内存中。因此，剩下的检测处理过程基本上都是在CPU和内存中执行的，其处理时间仅占整个文件系统检测时间的5%~10%。

在这个处理阶段中，fsck将会检查并分析文件系统中的所有目录，检测其中的目录项是否指向未分配的或坏的信息节点，同时也检测根目录对应的信息节点，删除前一阶段发现的含有无效信息节点号的目录项，报告并修复下列错误：

- 根目录信息节点中的模式（即访问权限）与状态字段是否有误；
- 目录信息节点中的地址指针是否超出文件系统的范围；
- 目录项中的信息节点号是否有效；
- 目录的完整性是否存在问题。

3. 第三阶段：检测目录的连接性

第三阶段主要检测目录的连接性。fsck将利用第二阶段缓存的信息，跟踪每一个目录直至到达根目录的整个路径。此时，fsck将会同时检测每个目录文件中的父目录项“..”，确认其合法有效。在跟踪目录的过程中，如果发现无法回溯到根目录的任何目录，fsck将会把这样的目录移至/lost+found目录。

在这个阶段，fsck将会根据第二阶段对目录的考察，重点检测信息节点表中是否存在没有目录归属的信息节点，因为每个已分配的信息节点至少都应有一个目录归属。报告并修复下列错误：

- 是否存在没有目录归属的信息节点;
- lost+found 目录是否存在, 或目录项位置是否已全部占用。

除了lost+found 目录下没有空间之外, 这个阶段发现的所有错误都是可以校正的(包括自动检测与修复方式)。

4. 第四阶段: 检测信息节点的链接计数

在第四阶段检测过程中, fsck将重点检测所有信息节点的链接计数。fsck将会遍历所有信息节点, 读取其中的链接计数, 然后使用第一阶段检测期间缓存的信息节点链接计数, 与第二和第三阶段计算出来的链接计数, 逐一进行比较。如果存在未删除的, 但链接计数为0的文件, 把这样的文件移至/lost+found 目录。报告并修复下列错误:

- 是否存在未引用的文件、符号链接文件与目录;
- 文件、目录、符号链接文件以及特殊文件的链接计数是否正确;
- 空闲信息节点的统计计数是否正确。

5. 第五阶段: 检测数据块组及文件系统的汇总信息

在最后一个检测阶段中, fsck将检测文件系统汇总信息的有效性。fsck将利用第一阶段构造的数据块及信息节点位图与文件系统实际位图进行比较。如果需要, 则对文件系统实际位图进行必要的校正。

在这个处理阶段, fsck主要检查数据块区, 检测坏块、重复的数据块、缺失的数据块、空闲和已用数据块计数、空闲和已用信息节点以及相应的位图, 报告并修复下列错误:

- 已用信息节点位图是否遗漏了已分配的信息节点;
- 已用信息节点位图中是否出现了空闲信息节点;
- 空闲数据块位图是否遗漏了空闲数据块;
- 空闲数据块的统计计数是否正确;
- 已用信息节点的统计计数是否正确。

一旦完成文件系统的检测, fsck命令将会在最后给出整个分析处理的总结报告, 即给出下列的文件系统使用情况汇总信息:

```
filesystem: used/max files (percent non-contiguous), used/max blocks
```

其中每个字段的意义表述如下:

- filesystem: 当前检测的文件系统的名字;
- used/max files: 当前已创建文件的数量/文件系统的最大文件容量;
- percent non-contiguous: 磁盘数据块的非线性化比率;
- used/max blocks: 当前已经占用的数据块数量/文件系统的最大数据块数量。

例如, 当以交互方式运行fsck命令结束之后, fsck将会给出下列类似的信息:

```
$ sudo fsck -t ext4 /dev/sdb1
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
/dev/sdb1 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
```

```
Pass 5: Checking group summary information
Inode bitmap differences: -12
Fix<y>? yes

/dev/sdb1: ***** FILE SYSTEM WAS MODIFIED *****
/dev/sdb1: 11/16000 files (9.1% non-contiguous), 7382/63924 blocks
$
```

上述汇总信息表示， /dev/sdb 存储设备对应的文件系统中共有187个文件， 占用了8750个数据块， 文件系统总共可以容纳16 000个文件， 拥有64 000个数据块， 数据块的非线性化比率为1.6%。

14.6 调试文件系统

14.6.1 概述

debugfs是一个交互式文件系统调试程序， 用于考察与修复Ext2/3/4文件系统， 恢复误删的文件等。其语法格式简写如下：

```
debugfs [-wci] [-b blocksize] [-s superblock] [-f cmd_file] [-R request] [device]
```

其中， device是Ext2/3/4文件系统的设备文件名。其他选项的说明如表14-6所示。

表14-6 debugfs命令的常用选项及简单说明

选项	简单说明
-w	以读写方式打开文件系统。如果未指定这个选项， 将会以只读方式打开文件系统。在文件系统调试期间发生的任何改动， 无法写入文件系统
-c	假定文件系统已经严重受损， 因而只能以非常规的只读方式打开文件系统
-i	表示指定的文件是一个采用e2image命令创建的Ext2/3/4文件系统映像文件。注意， 由于Ext2/3/4文件系统映像文件仅包含超级块、数据块组描述、数据块位图与信息节点位图， 以及信息节点表， 许多debugfs子命令（如ls和dump等）无法正确地执行
-b blocksize	强制使用指定的数据块大小作为文件系统逻辑数据块的大小， 而不是由mkfs命令根据磁盘分区的容量确定的逻辑数据块大小
-s superblock	使debugfs按给定的数据块号读取文件系统的超级块（而不是读取默认的主超级块）。注意，“-s”选项要求同时给出“-b”选项
-f cmd_file	以批处理的方式， 从指定的脚本文件cmd_file中读取并执行debugfs支持的子命令
-R request	使debugfs仅仅执行一个指定的请求， 然后就结束命令的执行

14.6.2 交互调试子命令

例如， 为了调试软盘上的Ext3文件系统， 可以使用下列命令：

```
$ sudo debugfs -w /dev/fd0
[sudo] password for gxqing:
debugfs 1.41.9 (22-Aug-2009)
debugfs:
```

debugfs命令提供一组丰富的交互式调试子命令，用于调试、修改文件系统。表14-7给出了部分常用的子命令及简单说明。

表14-7 debugfs支持的部分交互式调试子命令

交互式调试子命令	简单说明
bmap <i>filespec logical_block</i>	<p>显示指定信息节点中与指定逻辑数据块号相对应的物理数据块号。利用这个命令，可以找出一个文件的实际数据存储位置。例如，下列命令表示信息节点号2（即根目录）占用的第一个（0）逻辑数据块，其数据块号为504：</p> <pre>debugfs: bmap <2> 0 504 debugfs:</pre>
cat <i>filespec</i>	<p>显示指定文件的数据内容。例如，假定文件已经存在，或已利用write子命令，把/etc/hostname文件复制到当前正在调试的文件系统，其文件名也为hostname，信息节点号为14，下列命令将会显示出这个文件的内容：</p> <pre>debugfs: cat <14> iscas debugfs:</pre>
cd <i>filespec</i>	进入指定的目录，并把新的目录作为当前工作目录
close	关闭当前打开的文件系统
clri <i>file</i>	清除指定文件中的数据
dump [-p] <i>filespec out_file</i>	<p>把指定文件中的数据写入指定的输出文件。如果给出了“-p”选项，则按源文件信息节点中的属主、用户组和访问权限等属性设置输出文件。注意，输出文件位于调用debugfs命令之前所在的文件系统，而非当前正在调试的文件系统。因此，如果给出的是相对路径名，则输出文件应当是相对于运行debugfs命令时所处的工作目录</p>
expand_dir <i>filespec</i>	扩充指定目录文件的容量，即再为目录文件分配一个数据块
ffb [<i>count</i> [<i>goal</i>]]	<p>find_free_block子命令的简化形式，用于找出空闲的数据块，或从指定的数据块位置（goal）开始，找出指定数量（count）的空闲数据块地址（数据块号）。例如，下列命令表示文件系统中第一个空闲数据块的地址是281：</p> <pre>debugfs: ffb Free blocks found: 519 debugfs:</pre>
ffi [<i>dir</i> [<i>mode</i>]]	<p>find_free_inode子命令的简化形式，用于找出空闲的信息节点。如果以信息节点号的形式指定了某个目录（dir），则从指定的目录（信息节点号）开始找出空闲的信息节点。例如，下列命令表示文件系统中第一空闲的信息节点号为15：</p> <pre>debugfs: ffi Free inode found: 15 debugfs:</pre>
freeb <i>block</i> [<i>count</i>]	把指定的数据块标记为未分配的空闲数据块。如果给出了数据块计数，则从指定的数据块开始依次处理
freei <i>filespec</i>	释放指定的信息节点
help或?	输出debugfs命令支持的子命令列表。当进入debugfs命令的交互环境时，可以使用help列出debugfs命令支持的所有子命令

(续表)

交互式调试子命令	简单说明
<code>icheck blocks</code>	<p>输出占用了指定数据块的信息节点（列表）。例如，下列命令表示数据块7169、7681和518分别对应于信息节点12、13和14：</p> <pre> debugfs: icheck 7169 7681 518 Block Inode number 7169 12 7681 13 518 14 debugfs: </pre>
<code>imap filespec</code>	<p>输出指定文件信息节点在信息节点区中的位置。例如，根目录的信息节点2在信息节点区中的位置如下：</p> <pre> debugfs: imap <2> Inode 2 is part of block group 0 locate at block 254, offset 0x0080 debugfs: </pre>
<code>init_filesys device blocksize</code>	<p>在指定的设备上，使用指定的数据块容量，创建一个Ext2文件系统。注意，这种做法只是调用低级的库函数，设置超级块和数据块组描述而已，不会完全初始化所有的数据结构</p>
<code>kill_file filespec</code>	<p>释放指定文件的信息节点及其数据块。但这种文件删除方式并不能删除指定文件在目录文件中的目录项</p>
<code>lcd directory</code>	<p>把运行debugfs命令时的工作目录改换为系统中的指定目录（而非当前正在调试的文件系统中的目录）</p>
<code>logdump [-acs] [-b <block>]</code> <code>[-i <filespec>]</code> <code>[-f <journal_file>]</code> <code>[output_file]</code>	<p>转储Ext3文件系统日志信息。通常，日志的信息节点是由超级块指定的，但可以利用“-i”选项指定日志实际使用的信息节点。“-f”选项用于指定包含日志数据的文件。“-s”选项表示利用超级块中的备份信息确定日志的存储位置。“-a”选项表示输出数据块组描述所在的所有数据块中的内容。“-b”选项表示输出指定数据块中的所有日志记录。“-c”选项意味着输出“-a”和“-b”选项选择的所有数据块中的内容</p>
<code>ln filespec dest_file</code>	<p>使用指定的文件（filespec）创建一个硬链接文件（dest_file）。注意，这种做法不会调整文件的链接计数</p>
<code>ls [-l] [-d] [filespec]</code>	<p>列出指定目录中的文件。其中的“-l”选项表示以长列表的格式显示更多的文件属性信息。“-d”选项表示仅列出删除的目录项。例如，下列命令用于显示根目录中的文件列表：</p> <pre> debugfs: ls -l <2> 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 . 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 .. 11 40700 (2) 0 0 12288 11-Feb-2010 17:36 lost+found 12 100644 (1) 1000 1000 18562 11-Feb-2010 17:44 info 13 100644 (1) 1000 1000 10668 11-Feb-2010 17:45 memo 14 100644 (1) 1000 1000 6 11-Feb-2010 17:45 hostname </pre> <p>(END)</p>
<code>mi filespec</code>	<p>modify_inode子命令的简化形式，用于修改指定文件的信息节点数据结构中的数据。这是一种交互式操作，可以逐一审查系统提示的信息节点数据，必要时进行适当的修改</p>

(续表)

交互式调试子命令	简单说明
<code>mkdir <i>filespec</i></code>	<p>创建一个指定的目录, 例如, 下列命令将会在当前目录中创建一个newdir子目录:</p> <pre>debugfs: mkdir newdir debugfs: ls -l 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 . 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 .. 11 40700 (2) 0 0 12288 11-Feb-2010 17:36 lost+found 12 100644 (1) 1000 1000 18562 11-Feb-2010 17:44 info 13 100644 (1) 1000 1000 10668 11-Feb-2010 17:45 memo 14 100644 (1) 1000 1000 6 11-Feb-2010 17:45 hostname 15 40755 (2) 0 0 1024 11-Feb-2010 18:05 newdir</pre> <p>(END)</p>
<code>mknod <i>filespec</i> [p] [c b <i>major minor</i>]</code>	<p>创建一个指定的设备特殊文件 (如命名的管道文件、字符特殊文件或块特殊文件)</p>
<code>ncheck <i>inode_nums</i></code>	<p>输出指定信息节点号对应的文件名 (列表), 例如:</p> <pre>debugfs: ncheck 12 13 14 15 Inode Pathname 12 //info 13 //memo 14 //hostname 15 //newdir debugfs:</pre>
<code>open [-w] [-f] [-i] [-c] [-b <i>blocksize</i>] [-s <i>superblock</i>] <i>device</i></code>	<p>在进入debugfs交互命令环境之后, 打开准备调试的文件系统。其中的“-w”选项表示以写方式打开文件系统。“-f”选项表示强制打开文件系统, 即使文件系统存在问题, 无法正常打开文件系统。“-c”、“-b”、“-i”和“-s”具有与debugfs命令同名选项相同的功能</p>
<code>pwd</code>	<p>显示当前工作目录。例如, 下列命令表示debugfs命令当前处于正在调试的文件系统的根目录:</p> <pre>debugfs: pwd [pwd] INODE: 2 PATH: / [root] INODE: 2 PATH: / debugfs:</pre>
<code>quit或q</code>	退出debugfs命令
<code>rdump <i>directory destination</i></code>	<p>把指定目录中的所有文件 (包括普通文件、符号链接文件以及子目录等) 递归地复制到指定的目的目录位置 (位于非调试的文件系统中)。注意, 在使用这个子命令之前, 目的目录必须已经存在。例如, 为了把调试文件系统当前目录中Project子目录下的所有文件备份到/var/backup目录, 可以使用下列命令:</p> <pre>debugfs: rdump Project /var/backup debugfs:</pre>
<code>rm <i>pathname</i></code>	<p>删除指定的文件。如果删除文件时导致信息节点的链接计数为0, 则释放相应的信息节点。这个子命令的功能等同于unlink()系统调用</p>

（续表）

交互式调试子命令	简单说明
<code>rmdir filespec</code>	删除指定的目录，例如，下列命令表示删除新建的newdir 目录 debugfs: rmdir newdir debugfs:
<code>setb block [count]</code>	把指定的数据块标记为已分配的数据块。如果给出了数据块计数，则从指定的数据块开始，把指定数量的数据块均标记为已分配的数据块
<code>set_bg bgnum field value</code>	set_block_group子命令的简化形式，用于修改指定的数据块组描述，把其中的指定字段设置为指定的值。此外，还可以利用“-l”选项，列出set_block_group子命令能够设置的数据块组描述中的所有字段
<code>seti filespec</code>	把指定的信息节点标记为已分配的信息节点
<code>sif filespec field value</code>	set_inode_field子命令的简化形式，用于修改指定的信息节点，把其中的指定字段设置为指定的值。此外，还可以利用“-l”选项，列出set_inode_field子命令能够设置的所有信息节点字段
<code>ssv field value</code>	set_super_value子命令的简化形式，用于修复超级块，把其中的指定字段设置为指定的数值。此外，还可以利用“-l”选项，列出set_super_value子命令能够设置的所有超级块字段
<code>stat filespec</code>	show_inode_info子命令的简化形式，用于显示指定信息节点结构定义中的数据内容
<code>stats [-h]</code>	show_super_stats子命令的简化形式，用于显示超级块和数据块组描述中的数据。如果给出了“-h”选项，则仅显示超级块中的数据
<code>testb block [count]</code>	测试指定的数据块是否已经分配。如果同时给出了数据块计数，则从指定的数据块开始，测试指定数量的数据块是否均已分配。例如： debugfs: testb 7682 Block 7682 marked in use debugfs: testb 7800 Block 7800 not in use debugfs:
<code>testi filespec</code>	测试指定的信息节点是否已经分配，例如： debugfs: testi <2> Inode 2 is marked in use debugfs: testi <16> Inode 16 is not in use debugfs:
<code>undel <inode> [pathname]</code>	undelete子命令的简化形式，用于恢复删除的信息节点号，把指定的信息节点标记为当前正在使用的信息节点。同时，如果指定了路径名，则把恢复的信息节点链接到指定的目录。在使用undel命令恢复误删的文件之后，应当总是运行e2fsck命令，修复文件系统。注意，如果是恢复大量误删的文件，把信息节点连接到一个目录有可能需要扩展目录的存储空间，导致分配的数据块取自某个已删的文件，使相应的文件无法恢复。因此，比较安全的做法是在恢复所有的信息节点时，不指定目的目录。最后，再单独使用debugfs的ln子命令，把信息节点链接到一个目的目录，或使用e2fsck命令修复文件系统，把恢复的所有信息节点链接到lost+found目录。参见后面的讨论

(续表)

交互式调试子命令	简单说明
<code>unlink pathname</code>	删除指定的文件。注意，如果删除的文件是一个硬链接文件， <code>debugfs</code> 命令不会真正调整信息节点中的链接计数字段
<code>write source_file out_file</code>	把文件系统中指定文件（ <code>source_file</code> ）中的数据复制到新建的输出文件（ <code>out_file</code> ）。注意，这里的输出文件指的是调试文件系统中的文件。例如，为了把系统文件/ <code>etc/profile</code> 复制到当前正在调试的文件系统中，可以使用下列命令： <pre>debugfs: write /etc/fstab fstab Allocated inode: 15 debugfs: ls -l</pre> <pre> 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 . 2 40777 (2) 0 0 1024 11-Feb-2010 17:45 .. 11 40700 (2) 0 0 12288 11-Feb-2010 17:36 lost+found 12 100644 (1) 1000 1000 18562 11-Feb-2010 17:44 info 13 100644 (1) 1000 1000 10668 11-Feb-2010 17:45 memo 14 100644 (1) 1000 1000 6 11-Feb-2010 17:45 hostname 15 100644 (1) 0 0 752 11-Feb-2010 18:11 fstab </pre>
(END)	

在`debugfs`命令的语法格式中，子命令中的参数`filespec`用于指定文件系统中的信息节点或文件的路径名。在指定`filespec`参数时，可以采用两种形式：第一种形式是在信息节点号前后加单书名号，如<2>，第二种形式是采用常规的路径名。路径名的第一个字符如果是斜杠字符“/”，表示`debugfs`当前打开的文件系统的根目录，否则表示相对于`debugfs`当前工作目录的相对路径名。如果想要改换到不同的目录位置，可以使用`debugfs`的`cd`子命令。

若想显示超级块和数据块组描述中的当前数据，可以使用下列`stats (show_super_stats)`命令：

```

debugfs: stats
Filesystem volume name:      <none>
Last mounted on:            <not available>
Filesystem UUID:            38b06c28-54e0-42ff-b10f-c9bdc221ad2d
Filesystem magic number: 0xEF53
Filesystem revision #:       1 (dynamic)
Filesystem features:         has_journal ext_attr resize_inode dir_index ...
Filesystem flags:            signed_directory_hash
Default mount options:       (none)
Filesystem state:            clean
Errors behavior:             Continue
Filesystem OS type:          Linux
Inode count:                 16000
Block count:                 63924
Reserved block count:        3196
Free blocks:                 56492
Free inodes:                 15985
First block:                 1
Block size:                  1024
Fragment size:              1024
.....
:
```

在修复超级块时，可以利用“-l”选项列出ssv (set_super_value) 命令能够设置的超级块字段

```
debugfs: ssv -l
Superblock fields supported by the set_super_value command:
inodes_count          unsigned integer
blocks_count          unsigned integer
r_blocks_count        unsigned integer
free_blocks_count     unsigned integer
free_inodes_count     unsigned integer
first_data_block      unsigned integer
log_block_size        unsigned integer
log_frag_size         integer
blocks_per_group       unsigned integer
frags_per_group       unsigned integer
inodes_per_group      unsigned integer
mtime                date/time
wtime                date/time
mnt_count            unsigned integer
max_mnt_count        integer
state                unsigned integer
errors               unsigned integer
.....
:
```

若想显示信息节点数据结构中的数据内容，可以使用下列stat命令：

```
debugfs: stat <2>
Inode: 2   Type: directory   Mode: 0777   Flags: 0x0
Generation: 0   Version: 0x00000000
User:      0   Group:      0   Size: 1024
File ACL: 0   Directory ACL: 0
Links: 3   Blockcount: 2
Fragment: Address: 0   Number: 0   Size: 0
ctime: 0x4b73d1a9 -- Thu Feb 11 17:45:13 2010
atime: 0x4b73cf84 -- Thu Feb 11 17:36:04 2010
mtime: 0x4b73d1a9 -- Thu Feb 11 17:45:13 2010
BLOCKS:
(0):504
TOTAL: 1
(END)
```

14.6.3 恢复误删的文件

一提到文件系统调试器，自然会联想到在不经意删除了文件之后，是否能够恢复误删的文件？在一定的条件下，回答是肯定的，但并非总是如此。在使用GNOME桌面环境访问Linux系统时，同Windows系统一样，删除的文件和目录总是缓存在回收站中。如果发现误删了文件，可以非常方便地从回收站中恢复文件。

但在使用命令行界面访问Linux系统时，如果使用rm等命令删除文件，一经执行，便会立即释放文件的信息节点和数据块，回收站在此不起任何作用。如果误删了文件，仅当满足下列条件时，才能利用debugfs命令挽救删除的文件。

(1) 删除文件后, 系统或用户没有在文件系统中创建任何新的文件(包括目录、符号链接文件等各种文件)。这个条件表示因删除文件释放的信息节点和数据块没有再次分配使用。

(2) 删除文件后, 系统或用户也没有在文件系统中扩充原有的任何文件。这个条件表示删除文件释放的数据块没有再次分配使用。

如果满足上述两个条件, 可以立即卸载文件系统, 仿照下列步骤, 利用debugfs命令恢复误删的文件。

(1) 利用ffi (find_free_inode) 子命令, 找出第一个空闲的信息节点。

(2) 利用stat子命令, 检查信息节点数据结构中的BLOCKS字段(Ext2/Ext3文件系统)。如果其中的数据块指针数据为空, 说明当前的信息节点根本没有用过。如果其中给出了数据块号——单个数据块号或一个数据块号范围, 说明相应的信息节点是一个删除的文件。

(3) 利用cat子命令, 检查其中的数据是否为误删的文件。

(4) 如果是, 利用dump子命令, 把数据保存到系统中指定的文件。文件恢复过程结束。

(5) 否则, 利用seti子命令, 在信息节点位图中, 把指定的信息节点标记为已分配。

(6) 从步骤(1)开始重复执行上述过程。

假定1.44MB软盘的Ext3文件系统中存在下列文件, 在操作过程中不慎误删了其中的重要文件security:

```
$ sudo mount /dev/fd0 /mnt
[sudo] password for gqxing:
$ ls -l /mnt
total 71
drwx----- 2 root root 12288 Feb 11 13:02 lost+found
-rw-r--r-- 1 gqxing gqxing 21822 Feb 11 13:03 schedule
-rw-r--r-- 1 gqxing gqxing 10560 Feb 11 13:03 security
-rw-r--r-- 1 gqxing gqxing 23666 Feb 11 13:03 typescript
$ rm /mnt/security
$
```

且由于发现及时, 对文件系统没有再做其他操作, 故文件系统满足文件恢复的要求。此时可以立即卸载文件系统, 调用debugfs命令, 执行文件的恢复过程:

```
$ sudo umount /mnt
$ sudo debugfs -w /dev/fd0
debugfs 1.41.9 (22-Aug-2009)
debugfs: ls -l
      2  40777 (2)      0      0    1024 11-Feb-2010 13:28 .
      2  40777 (2)      0      0    1024 11-Feb-2010 13:28 ..
     11  40700 (2)      0      0   12288 11-Feb-2010 13:02 lost+found
     12 100644 (1)   1000   1000   21822 11-Feb-2010 13:03 schedule
     14 100644 (1)   1000   1000   23666 11-Feb-2010 13:03 typescript

(END)
debugfs: ffi
Free inode found: 13
debugfs: stat <13>
Inode: 13   Type: regular   Mode: 0644   Flags: 0x0
Generation: 507723457   Version: 0x00000000
User: 1000   Group: 1000   Size: 10560
```

```

File ACL: 0      Directory ACL: 0
Links: 0      Blockcount: 22
Fragment:  Address: 0      Number: 0      Size: 0
ctime:        0x4b739573  -- Thu Feb 11 13:28:19 2010
atime:        0x4b738fb7  -- Thu Feb 11 13:03:51 2010
mtime:        0x4b738fb7  -- Thu Feb 11 13:03:51 2010
dtime:        0x4b739573  -- Thu Feb 11 13:28:19 2010
BLOCKS:
(0-10):70-80
TOTAL: 11

(END)
debugfs:  cat <13>
# This is a testing file for debugfs
#
# There is a lot of important data in this file
.....
debugfs:  dump <13> rescued
debugfs:  quit
$ ls -l rescued
-rw-r--r-- 1 root root 10560 Feb 11 13:32 rescued
$ cat rescued
# This is a testing file for debugfs
#
# There is a lot of important data in this file
.....
$

```



恢复的文件rescued位于调用debugfs命令之前用户所在的工作目录，而非当前调试的文件系统，且由其输出数据可见，恢复后的文件rescued与原文件security的内容完全相同。

14.6.4 恢复误删的文件（续）

此外，如果利用unde1命令恢复误删的文件，其处理过程实际上相对更简单。例如，假定使用rm子命令误删了文件security，之前已知道其信息节点号为13（否则可以利用free1等命令找出其信息节点），可以利用下列unde1子命令，把误删的文件恢复到newdir目录中：

```

debugfs:  ls -l
      2  40777 (2)      0      0      1024 11-Feb-2010 16:08 .
      2  40777 (2)      0      0      1024 11-Feb-2010 16:08 ..
     11  40700 (2)      0      0     12288 11-Feb-2010 13:02 lost+found
     12 100644 (1)    1000    1000    21822 11-Feb-2010 13:03 schedule
     13 100644 (1)    1000    1000    10560 11-Feb-2010 16:08 security
     14 100644 (1)    1000    1000    23666 11-Feb-2010 16:08 typescript
     15  40755 (2)    1000    1000     1024 11-Feb-2010 16:08 newdir

(END)
debugfs:  rm security
debugfs:  ls -l
      2  40777 (2)      0      0      1024 11-Feb-2010 16:08 .

```

```

      2  40777 (2)      0      0    1024 11-Feb-2010 16:08 ..
     11  40700 (2)      0      0   12288 11-Feb-2010 13:02 lost+found
     12 100644 (1)    1000    1000   21822 11-Feb-2010 13:03 schedule
     14 100644 (1)    1000    1000   23666 11-Feb-2010 16:08 typescript
     15  40755 (2)    1000    1000    1024 11-Feb-2010 16:08 newdir

(END)
debugfs: undel <13> newdir
debugfs: ls -l newdir
      15  40755 (2)    1000    1000    1024 11-Feb-2010 16:08 .
       2  40777 (2)      0      0    1024 11-Feb-2010 16:08 ..
      13 100644 (1)    1000    1000   10560 11-Feb-2010 16:08 <13>

(END)
debugfs: cat newdir/<13>
# This is a testing file for debugfs
#
# There is a lot of important data in this file
.....
debugfs:

```

undel子命令将会以信息节点号的形式“<inode>”命名恢复的文件。因此，在退出debugfs命令之后，可以安装文件系统，重新命名恢复的文件：

```

debugfs: quit
$ sudo mount /dev/fd0 /mnt
[sudo] password for gqxing:
$ cd /mnt/newdir
$ ls -l
total 11
-rw-r--r-- 1 gqxing gqxing 10560 Feb 11 16:08 <13>
$ mv \<13\> security
$ ls -l
total 11
-rw-r--r-- 1 gqxing gqxing 10560 Feb 11 16:08 security
$ cd
$ sudo umount /mnt
$

```

当删除的文件比较多，目录项的字符数量超过一个目录的现有容量时，可以采用另外一种文件恢复方式——在使用undel子命令恢复文件时，不要指定目的目录。具体做法是首先使用下列undel子命令，逐一恢复每个文件的信息节点：

```
undel <inode>
```

然后再用下列ln子命令，把每个文件逐一链接到指定的目录：

```
ln <inode> directory
```

最后再安装文件系统，重新命名恢复的所有文件。

采取上述方法的好处是，在恢复了所有信息节点之后，再把文件连接到一个目录，即使目录项超出指定目录现有的存储容量，因而需要申请新的数据块时，也不会占用已恢复文件的数据块。但缺点是需要针对欲恢复的文件，逐一运行undel和ln子命令。

还有一种简单的做法是借助fsck命令，实现文件的快速恢复。具体做法是：在运行undel子命令之后，修改文件系统超级块的状态标志，接着退出debugfs命令，最后再运行fsck命令。

假定当前调试的软盘文件系统中存在exports、fstab、group、passwd和sudoers5个重要文件。

```
$ sudo debugfs -w /dev/fd0
debugfs 1.41.9 (22-Aug-2009)
debugfs: ls -l
      2  40777 (2)      0      0    1024 11-Feb-2010 17:11 .
      2  40777 (2)      0      0    1024 11-Feb-2010 17:11 ..
     11  40700 (2)      0      0   12288 11-Feb-2010 17:09 lost+found
     12 100644 (1)    1000    1000     577 11-Feb-2010 17:10 exports
     13 100644 (1)    1000    1000     752 11-Feb-2010 17:10 fstab
     14 100644 (1)    1000    1000     953 11-Feb-2010 17:10 group
     15 100644 (1)    1000    1000    1960 11-Feb-2010 17:11 passwd
     16 100440 (1)      0      0     557 11-Feb-2010 17:11 sudoers

(END)
debugfs:
```

后来由于某种原因遭到误删。此时，可以利用debugfs的undel与ssv (set_super_value) 子命令，以及fsck等命令，恢复误删的文件。具体步骤如下：

```
debugfs: rm exports
debugfs: rm fstab
.....
debugfs: ls -l
      2  40777 (2)      0      0    1024 11-Feb-2010 17:11 .
      2  40777 (2)      0      0    1024 11-Feb-2010 17:11 ..
     11  40700 (2)      0      0   12288 11-Feb-2010 17:09 lost+found

(END)
debugfs: undel <12>
debugfs: undel <13>
debugfs: undel <14>
debugfs: undel <15>
debugfs: undel <16>
debugfs: ssv state 200
debugfs: quit
$ sudo fsck -t ext3 /dev/fd0
fsck from util-linux-ng 2.16
e2fsck 1.41.9 (22-Aug-2009)
/dev/fd0 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 12
Connect to /lost+found<y>? yes

Inode 12 ref count is 2, should be 1. Fix<y>? yes

Unattached inode 13
Connect to /lost+found<y>? yes
```

```

Inode 13 ref count is 2, should be 1.  Fix<y>? yes
Unattached inode 14
Connect to /lost+found<y>? yes

Inode 14 ref count is 2, should be 1.  Fix<y>? yes
Unattached inode 15
Connect to /lost+found<y>? yes

Inode 15 ref count is 2, should be 1.  Fix<y>? yes
Unattached inode 16
Connect to /lost+found<y>? yes

Inode 16 ref count is 2, should be 1.  Fix<y>? yes

Pass 5: Checking group summary information
/dev/fd0: ***** FILE SYSTEM WAS MODIFIED *****
/dev/fd0: 16/184 files (0.0% non-contiguous), 53/1440 blocks
$ sudo mount /dev/fd0 /mnt
$ sudo ls -l /mnt/lost+found
total 6
-rw-r--r--  1 gqxing  gqxing  577 Feb 11 17:10 #12
-rw-r--r--  1 gqxing  gqxing  752 Feb 11 17:10 #13
-rw-r--r--  1 gqxing  gqxing  953 Feb 11 17:10 #14
-rw-r--r--  1 gqxing  gqxing 1960 Feb 11 17:11 #15
-r--r----- 1 root    root    557 Feb 11 17:11 #16
$

```

最后，可以根据文件的内容，重新命名#12至#16五个文件。注意，在上述步骤中，利用ssv (set_super_value) 命令修改超级块中的文件系统状态字段是至关重要的，其目的是把文件系统的超级块置于非正常状态，否则fsck不会修复文件系统。

第15章 系统启动与关机

Linux系统的启动过程是一个复杂的内部引导过程。本章将从磁盘分区与GRUB引导程序开始，讨论系统的引导与生成过程，最后介绍系统的关机。掌握系统启动过程的工作原理，不仅有助于理解与修复系统启动过程中出现的问题，也有助于了解从何处入手，定制应用程序的启动脚本。

Linux系统的启动过程涉及磁盘分区、加电引导、运行初始引导程序以及Linux系统引导程序等一系列复杂的引导与内部处理过程。在加电自检之后，计算机开始加载并执行系统盘0号扇区中的初始引导程序，确定引导哪一个系统，然后加载并启动操作系统提供的引导程序，检测系统的硬件配置，加载操作系统内核，执行系统初始化，设置硬件运行环境，加载必要的软件模块，组织系统的内部数据结构，运行系统启动脚本，建立多用户运行环境，直至最终完成系统的启动，其间经过一系列复杂的处理过程。

下面以Ubuntu Linux在Intel x86 硬件系统上的实现为例，从磁盘分区和GRUB引导程序开始，介绍系统的引导与生成过程，从而说明系统的整个启动过程，最后简单介绍系统的关机。

15.1 磁盘分区与GRUB

在Intel x86 体系结构的计算机系统中，系统磁盘的划分如图1 5-1 所示。其中，位于0 号扇区512 字节的数据块为主引导记录（Master Boot Record，MBR）。GRUB使用0 号扇区存储初始引导程序（bootimg）。紧随0 号磁道之后的1 ~62 号扇区（31 KB）可以用做核心引导程序（core.img）的存储区。从63 号扇区开始是操作系统分区。

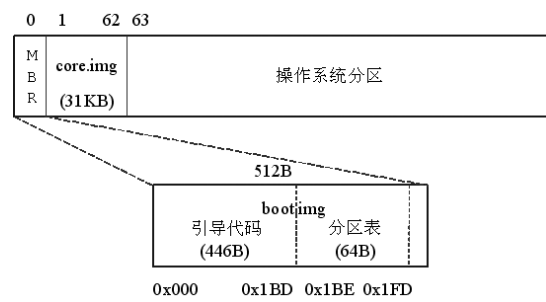


图1 5-1 系统磁盘布局

15.1.1 磁盘分区

512 个字节的主引导记录可以分为三部分，其中前446 个字节为计算机系统的初始引导程序，也称第一阶段引导程序。中间的64 个字节为磁盘分区表，用于描述4 个操作系统的分区信息，包括分区引导标志（说明是否可以引导）、每个操作系统分区的起始位置与容量，以及分区的系统类型。最后一部分只占2 个字节，用于存储主引导记录扇区的标志代码，其中的0xAA55 表示当前的0 号扇区即为主引导记录，如图1 5-2 所示。

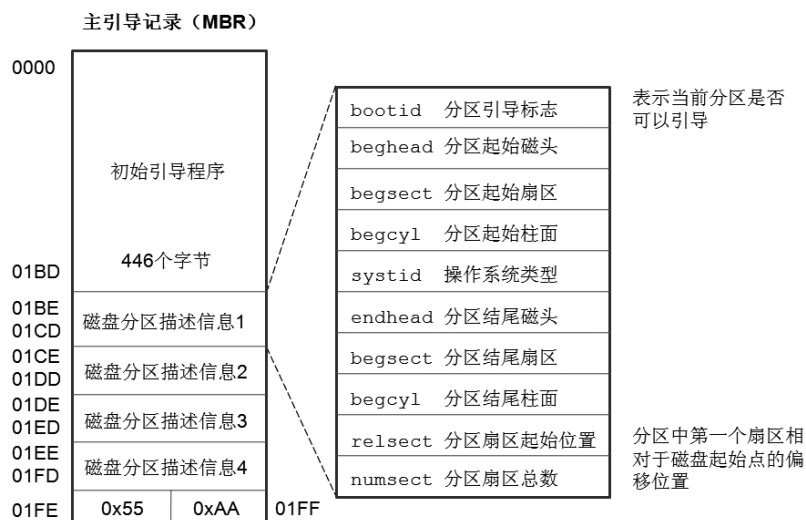


图1 5-2 系统硬盘主引导块

在磁盘分区表中，如果分区引导标志为0x0，说明相应的分区为非活动分区；如果引导标志为0x80，说明相应的磁盘分区是可引导的。至于究竟启动哪一个分区中的操作系统，取决于分区表中每个分区的引导标志，或用户在初始启动过程中做出的选择。操作系统类型字段表示相应磁盘分区的系统性质，如0x82表示Linux swap分区，0x83表示Linux引导分区，0x85表示Linux扩展分区，0x8e表示Linux LVM分区，0x07表示Windows NTFS分区，0x0b表示Win95 FAT32分区，以及0x0c表示Win95 FAT32（LBA）分区等。

如果系统中仅仅安装了一个Ubuntu Linux操作系统，即除了0号磁道之外，其他所有的存储空间均为Linux系统分区，其典型的磁盘空间布局结构如图1 5-3所示（假定安装Ubuntu Linux系统时采用的是默认磁盘分区布局，即没有额外划分其他文件系统分区）。

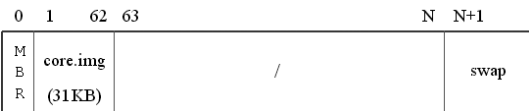


图1 5-3 Linux系统磁盘分区布局结构（1）

如果计算机中安装了多个系统，如安装Windows之后又安装了Ubuntu Linux系统，其系统磁盘分区布局的典型结构如图1 5-4所示（假定安装Ubuntu Linux系统时采用的是默认磁盘分区布局，即没有额外划分其他文件系统分区）。



图1 5-4 Linux系统磁盘分区布局结构（2）

15.1.2 GRUB引导程序

Ubuntu Linux采用GRUB作为引导程序。GRUB（Grand Unified Bootloader，统一引导管

理器)是一个通用引导程序,使用户能够在引导系统时选择启动的操作系统,或选择同一操作系统的不同版本。GRUB是目前应用最广泛的引导程序,许多Linux系统现均采用GRUB作为默认的引导程序。

GRUB提供三个用户界面,每个界面都允许用户直接引导操作系统,在系统启动期间,也可以在三个GRUB界面之间切换。

第一个是引导菜单界面,也是采用GRUB引导程序时默认的系统引导菜单界面。当安装完毕重新引导系统时,引导菜单界面通常会出现于屏幕上,利用上下箭头键可以选择需要引导的系统。如果长时间没有按键输入,GRUB会自动引导默认的操作系统。

第二个是引导菜单编辑界面。当出现操作系统引导菜单时,输入字母“e”便可以进入菜单编辑器。此时,可以临时修改系统引导菜单。例如,输入字母“o”可以在当前行后面增加引导项,输入大写字母“O”可以在当前行前面增加引导项,输入字母“d”可以删除引导项,输入字母“e”可以编辑引导项等。修改后可以按Enter键确认,按B键开始引导系统或按Esc键取消修改,重新回到GRUB引导菜单。



在编辑系统引导菜单期间所做的任何修改都是临时性的,在下次引导时,这些修改将会消失(如果想要永久修改引导菜单,需要编辑/boot/grub/grub.cfg文件)。在测试新编译的系统内核时,菜单编辑界面是比较有用的。

第三个是命令行界面,在操作系统引导菜单中按下C键,便可进入命令行界面。命令行是最基本的GRUB界面,也是最灵活的界面。命令行界面是一个小型的Shell环境,具有Bash的部分功能特性。此时可以输入GRUB命令,按下Enter键即可执行。

GRUB采用下列形式命名存储设备及其分区:

`(<device-type><device-number>,<partition-number>)`

其中,<device-type>表示设备类型。最常用的两个设备类型是hd(表示IDE磁盘)、sd(表示SCSI磁盘或USB移动盘)、ed(表示ESDI磁盘)、xd(表示XT磁盘)和fd(表示3.5英寸软盘)。其他设备类型还有nd(表示网络)和cd(表示CD/DVD)等。

<device-number>是BIOS能够识别的设备号,磁盘设备从0开始编号。第一个磁盘设备的编号为0,第二个磁盘设备的编号为1,依次类推。例如,GRUB把第一个IDE磁盘称做hd0,第二个IDE磁盘称做hd1。这种设备命名与编号方式类似于Linux系统早期内核采用的设备命名方式。例如,Linux系统内核使用的设备名/dev/hda等价于GRUB的hd0,/dev/hdb等价于hd1。

<partition-number>用于指定设备分区的编号。与先前版本的GRUB不同,在GRUB 2中,设备分区从1开始编号。例如,GRUB 2把第一个磁盘的第一个分区称做“(hd0,1)”,第二个磁盘的第三个分区称做“(hd1,3)”等。

实际上,在命名和引用设备及其分区时,GRUB并不区分IDE和SCSI等磁盘,所有硬盘均以hd命名。当指定整个磁盘,且不考虑其分区时(如需要将GRUB安装到一个磁盘的主引导记录时),只需将逗号“,”和分区号去掉即可。

15.1.3 GRUB配置文件

Ubuntu 9.10 Linux系统采用最新版的GRUB 2作为引导程序,其配置文件可以分为三种:/etc/default/grub文件、/etc/grub.d目录中的文件及/boot/grub/grub.cfg文件。现分述如下。

1. /etc/default/grub 文件

/etc/default/grub 文件是由 grub-set-default 命令生成的, 其中含有一组通用的配置选项, 用于设置 GRUB 运行环境, 生成 /boot/grub/grub.cfg 配置文件。其中 (若想取消某个变量设置, 可在变量名之前增加一个注释符号 “#”):

- GRUB_DEFAULT 用于指定 GRUB 引导的默认操作系统。引导菜单项从 0 开始编号, 0 表示第一个操作系统引导菜单项, 1 表示第二个操作系统引导菜单项, 依次类推。例如, GRUB_DEFAULT=0 表示引导第一个操作系统; GRUB_DEFAULT=saved 表示使用上一次引导期间选择的操作系统作为默认选项, 且在显示引导菜单时, 高亮显示先前选择的引导菜单项。同时, 把本次选择作为下一次启动时的默认引导菜单项。
- GRUB_TIMEOUT=number —— 用于指定开始引导之前的等待时间, 以便用户有机会选择引导不同的操作系统。例如, GRUB_TIMEOUT=10 表示等待时间为 10 秒。如果设置为 -1, 表示取消超时控制。在此情况下, GRUB 将会一直显示引导菜单, 直至用户做出选择。
- GRUB_HIDDEN_TIMEOUT=number —— 表示在引导期间禁止显示操作系统选择菜单, 除非在此变量之前增加一个注释符号 “#”。这个配置变量的默认值取决于计算机系统中装有几个操作系统。如果存在多个可引导的操作系统, 默认的动作是显示系统引导菜单 (意味着此变量之前存在一个注释符号), 否则默认的动作是不显示系统引导菜单。如果变量值大于 0, GRUB 将会在正式引导之前暂停指定的时间, 但不显示系统引导菜单, 超时后立即引导默认的操作系统。如果变量值为 0, GRUB 既不显示系统引导菜单, 也不会延迟系统引导。但在初始引导期间, GRUB 将会检测 Shift 键的状态, 如果无法确定 Shift 键的状态, GRUB 将会提供一个短暂的延迟时间, 只要及时按下 Esc 键, 仍会显示系统引导菜单。因此, 不管是否延迟, 如果不显示系统引导菜单, 只要按下 Shift 键或 Esc 键, 仍可看到标准的 GRUB 系统引导菜单界面。
- GRUB_HIDDEN_TIMEOUT_QUIET=true/false —— 如果这个变量为 true, 屏幕将是空的, GRUB 不会显示倒计时。如果为 false, GRUB 将会按照 GRUB_HIDDEN_TIMEOUT 变量指定的时间在空屏幕上显示一个计数器。
- GRUB_CMDLINE_LINUX=arg-string —— 如果指定了任何参数, GRUB 将会把这些参数附加到常规引导菜单项和恢复模式菜单项的 linux 配置参数后面。
- GRUB_CMDLINE_LINUX_DEFAULT=arg-string —— 表示把指定的参数附加到 linux 配置参数的后面。如果想在整个屏幕上滚动输出引导信息, 可以删除默认的参数 “quiet splash”。如果仅想在 Ubuntu 徽标下方显示有限的信息, 可以单独选用 “splash”。
- GRUB_TERMINAL=console —— 表示采用图形终端显示系统引导菜单 (仅使用于 grub-pc)。
- GRUB_GFXMODE=resolution —— 用于设置屏幕的分辨率, 如 1024×768 (4:3) 或 1280×800 (16:10) 等。
- GRUB_DISABLE_LINUX_UUID=true —— 表示是否允许 GRUB 把形如 “root=UUID=xxx” 的参数传递给 Linux 系统内核。
- GRUB_DISABLE_LINUX_RECOVERY=true —— 表示是否在引导菜单中增加一个系统恢复模式引导项。

- GRUB_DISABLE_OS_PROBER=true——表示是否允许os-prober脚本检测其他操作系统（如Windows）分区。

下面是Ubuntu 9.10 Linux系统提供的原始/etc/default/grub文件。

```
$ cat /etc/default/grub
# If you change this file, run 'update-grub' afterwards to update
# /boot/grub/grub.cfg.

GRUB_DEFAULT=0
#GRUB_HIDDEN_TIMEOUT=0
GRUB_HIDDEN_TIMEOUT_QUIET=true
GRUB_TIMEOUT=10
GRUB_DISTRIBUTOR=`lsb_release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
GRUB_CMDLINE_LINUX=""

# Uncomment to disable graphical terminal (grub-pc only)
#GRUB_TERMINAL=console

# The resolution used on graphical terminal
# note that you can use only modes which your graphic card supports via VBE
# you can see them in real GRUB with the command `vbeinfo'
#GRUB_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux
#GRUB_DISABLE_LINUX_UUID=true

# Uncomment to disable generation of recovery mode menu entries
#GRUB_DISABLE_LINUX_RECOVERY="true"

$
```

2./etc/grub.d目录中的文件

这个目录包含一组脚本文件，用于检索计算机中可引导的操作系统，创建相应的系统引导菜单项，自动生成/boot/grub/grub.cfg配置文件。脚本文件是按序命名的，文件的顺序决定了其输出结果在grub.cfg配置文件中出现的顺序。

在运行update-grub或grub-mkconfig命令时，将会按照脚本文件的序号，依次执行每一个脚本文件，把每个脚本文件的输出数据顺序写入/boot/grub/grub.cfg配置文件。如果想要增加自己的设置，可以修改40_custom文件，也可以在/etc/grub.d目录中创建一个新的脚本文件。注意，新建文件必须具有“可执行”的访问权限。

在Ubuntu 9.10 Linux系统中，/etc/grub.d目录含有下列脚本文件。

- 00_header——用于读取/etc/default/grub文件中的环境变量设置，如超时、引导菜单可见性及默认的系统引导项等。
- 05_debian_theme——用于设置文字颜色、背景颜色及主题等。例如，如果想要改变引导菜单的文字（白）和背景（蓝）颜色，以及选中菜单项的文字（蓝）和背景（白）颜色，可以分别设置下列变量：

```
set menu_color_normal=white/blue
set menu_color_highlight=blue/white
```

- 10_linux——用于确定系统内核的存储位置。

- 20_memtest86+——用于检索/boot/memtest86+bin文件。如果存在, 这个脚本文件将会构造一个内存测试菜单项, 即在GRUB引导界面中增加一个memtest86内存测试菜单项。
- 30_os-prober——用于检索系统中是否存在其他可引导的操作系统, 如Windows。如果存在, 这个脚本文件将会构造一个系统引导菜单项。
- 40_custom——是一个脚本模板, 可以利用这个文件增加自己的系统引导菜单项, 以便update-grub或grub-mkconfig命令能够自动插入grub.cfg配置文件。

例如, 如果想要增加一个引导菜单项, 用于启动自己定制的系统内核, 可以在40_custom文件中增加下列内容:

```
echo "Adding my new kernel ....." >&2
cat << EOF
menuentry "Ubuntu, linux 2.6.31-16-mykernel" {
    insmod ext2
    set root=(hd0,2)
    linux /boot/vmlinuz-2.6.31-16-mykernel
    root=/dev/sda2 ro quiet locale=zh_CN.UTF-8
    initrd /boot/initrd.img-2.6.31-16-generic
}
EOF
```

3./boot/grub/grub.cfg文件

GRUB 2引导程序本身使用的默认配置文件为/boot/grub/grub.cfg, 用于组织操作系统引导菜单, 以便GRUB在初始引导期间显示一个系统引导菜单界面, 使用户能够选择引导不同的操作系统。

除了带有注释符号“#”的注释行之外, grub.cfg配置文件由若干节组成, 每一节均以“### BEGIN filename ###”开始, 以“### END filename ###”结束, 其内容来自相应脚本文件的输出。其中:

- 00_header节主要用于加载/etc/default/grub文件中的全局变量设置。例如, 控制GRUB在用户选择启动哪一个操作系统之前等待多长时间再自动引导默认的系统, 如“set timeout=10”。指定默认引导菜单项的序号, 如“set default=0”。指定需要加载和执行的引导程序模块, 如“insmod ext2”等。
- 05_debian_theme节主要用于设置屏幕背景和文字的颜色, 例如:

```
set menu_color_highlight=black/white
set menu_color_normal=white/black
```

- 10_linux节主要用于设置Linux系统引导菜单, 指定系统内核和初始磁盘内存映像文件的存储位置、引导项以及需要加载的辅助系统引导程序等。
- 20_memtest86+节主要用于设置内存测试菜单, 指定内存测试程序等。
- 30_os-prober节主要用于检索其他操作系统, 如Windows系统, 设置GRUB引导菜单项。
- 40_custom节是一个模板, 用于增加用户特定的引导菜单项, 以便在执行update-grub等命令时能够把其输出数据插入grub.cfg配置文件。

在grub.cfg配置文件中, 每一节用于定义一个系统引导菜单, 每个引导菜单项以“menuentry 'title' {”开始, 直至右花括号结束。其中, title是系统引导菜单项的名字, 如“Ubuntu 9.10,

Linux 2.6.31-16-generic”。花括号中间的命令或配置参数告诉GRUB，在选定想要引导的系统之后，需要加载哪一个系统内核与磁盘内存映像文件，以及使用哪一些引导选项等。系统引导菜单项从0开始编号，0表示第一个系统引导菜单项，1表示第二个系统引导菜单项，依次类推。

实际上，grub.cfg配置文件相当于一个GRUB Shell脚本文件，其中可以包括任何GRUB命令或配置参数，也可以使用echo语句、test语句（包括简化的test语句），以及if-then-else等结构语句。

grub.cfg配置文件是由update-grub或grub-mkconfig命令自动生成的，其中的内容来自/etc/default/grub文件，以及/etc/grub.d目录中各个脚本文件的输出结果。因此，通常不应直接修改grub.cfg配置文件。否则，一旦再次运行update-grub或grub-mkconfig命令，将会重写此文件，导致附加的内容被覆盖。

表15-1给出了部分GRUB命令、配置参数及其简单说明，可用于配置grub.cfg文件。

表15-1 GRUB命令或配置参数

命令或配置参数	简单说明
blocklist <i>file</i>	<p>用于找出存储文件的实际磁盘扇区位置，其输出格式为“offset+length, offset[pos1, pos2]”。例如：</p> <pre>sh:grub> blocklist (hd0,2)/boot/grub/ext2.mod 5844592+10,5844602[0-424] sh:grub></pre> <p>上述输出信息表示ext2.mod文件位于5844592~5844602共11个扇区中，其中最后一个扇区仅存有424个字符（文件结束标志除外）</p>
boot	<p>用于引导操作系统内核。在发布boot命令之前，首先需要指定系统内核和必要的模块文件，然后再输入boot命令。例如：</p> <pre>sh:grub> chainloader (hd0)+1 sh:grub> boot sh:grub></pre>
cat <i>file</i>	<p>用于显示指定文本文件的内容。例如：</p> <pre>sh:grub> cat /boot/grub/device.map (hd0) /dev/sda sh:grub></pre>
chainloader <i>file</i>	<p>使用指定的文件作为下一个级联引导程序。如果需要加载的文件位于指定磁盘分区的第一个扇区中，可以采用数据块编号“+1”作为文件名，指定加载的起始数据块号，表示使用该分区第一个扇区中的内容作为引导程序，例如：</p> <pre>sh:grub> chainloader +1 sh:grub></pre> <p>如果想要查询可用的操作系统引导分区，可以使用下列命令：</p> <pre>sh:grub> chainloader (hd0,<tab> Possible partitions are: Partition hd0,1: Filesystem type ntfs, UUID cc485b1848485b011e Partition hd0,2: Filesystem type ext2, Last modification time 2009-11-06 15:47:16 Friday, UUID 472aa772-7de1-4b75-8c49-706973e9a213 Partition hd0,3: Unknown filesystem sh:grub> chainloader (hd0,</pre>

(续表)

命令或配置参数	简单说明
<code>configfile file</code>	用于指定替代/boot/grub/grub.cfg的其他配置文件
<code>default</code>	在GRUB 2中, <code>default</code> 只是一个变量, 用于指定引导的默认操作系统。如“ <code>set default=0</code> ”, 表示默认的系统引导菜单项为grub.cfg文件中定义的第一个操作系统
<code>halt</code>	用于立即关机
<code>help [cmdname]</code>	用于查询可用的GRUB命令
<code>initrd file</code>	用于指定引导期间需要加载的初始磁盘内存映像文件, 如/boot/initrd.img-version-generic文件。这是一个经由gzip命令压缩的“/”文件系统映像, 用于提供临时的“/”文件系统。系统内核可以利用这个虚拟的“/”文件系统, 加载必要的驱动程序模块, 在不安装任何物理磁盘的情况下完成系统的初始化。当GRUB引导程序把磁盘映像文件加载到内存之后, 系统内核会解压磁盘映像文件, 然后以只读或读写方式安装一个临时的虚拟“/”文件系统。在一个无盘的嵌入式Linux系统中, <code>initrd</code> 可以作为“/”文件系统继续使用
<code>insmod module</code>	用于加载或插入指定的模块。例如, 为了引导Ext2/3/4文件系统系统中的系统, 需要提前使用下列命令: <pre>sh:grub> insmod ext2 sh:grub></pre>
<code>linux file [args]</code>	用于指定需要引导的系统内核文件及其参数, 以便从指定的文件中加载系统内核映像。指定的文件名必须是root配置参数指定文件系统分区中的一个绝对路径名, 如/boot/vmlinuz-version-generic。args是需由GRUB引导程序传递给系统内核的命令行参数, 如“ <code>root=UUID=…… ro locale=zh_CN quiet splash</code> ”, 其中的UUID是“(hdX,Y)”的另外一种表示形式
<code>loopback [-d -p] device file</code>	使用指定映像文件作为引导设备。例如, 下列命令表示把第一个磁盘第二个分区中的CD/DVD映像文件作为引导设备: <pre>loopback loop (hd0,2)/iso/some.iso linux (loop)/boot/vmlinuz-2.6.31-14-generic initrd (loop)/boot/initrd.img-2.6.31-14-generic</pre>
<code>ls [-l -h -a] [file]</code>	用于显示计算机中装配的存储设备, 或显示指定目录中的文件。例如: <pre>sh:grub> ls (hd0) (hd0,4) (hd0,3) (hd0,2) (hd0,1) (fd0) sh:grub> ls -l /boot DIR 20091107114836 grub/ 1664737 20091016180349 System.map-2.6.31-14-generic 629174 20091016180349 abi-2.6.31-14-generic 111371 20091016180349 config-2.6.31-14-generic 128796 20091023161125 memtest86+.bin 1196 20091016180612 vmcoreinfo-2.6.31-14-generic 3890400 20091016180349 vmlinuz-2.6.31-14-generic 7645280 20091016183449 initrd.img-2.6.31-14-generic sh:grub></pre>
<code>lsmod</code>	用于显示内置或在使用前必须加载的外部模块, 例如: <pre>sh:grub> lsmod Name Ref Count Dependencies minicmd 1 ls 1 normal,extcmd ext2 4 fshelp sh:grub></pre>

（续表）

命令或配置参数	简单说明
<code>module file [args]</code>	用于加载系统引导期间需要用到的附加模块。选用的参数将会传递给模块，作为模块的命令行参数。注意，在加载任何模块之前必须先加载系统内核映像
<code>reboot</code>	重新引导计算机系统
<code>rescue</code>	进入系统维护模式
<code>root</code>	在GRUB 2中， <code>root</code> 是一个变量，用于指定系统内核所在的磁盘分区，如“ <code>set root = (hd0,2)</code> ”，表示系统内核位于第一个磁盘的第二个分区。如果/ <code>boot</code> 目录位于单独的磁盘分区，即存在单独的/ <code>boot</code> 文件系统， <code>root</code> 变量指定的磁盘分区应为/ <code>boot</code> 文件系统分区
<code>search [-f -l -u] name</code>	在可安装的所有文件系统分区中，按指定的文件、文件系统卷标或UUID等方式检索存储设备，例如： <pre>sh:grub> search -f /boot/grub/grub.cfg hd0,2 sh:grub></pre>
<code>set [envar=value]</code>	用于显示或设置环境变量。例如： <pre>sh:grub> set ?=16 color_highlight= default=0 gfxmode=640x480 have_grubenv=true menu_color_highlight=black/white menu_color_normal=white/black pager= prefix=(hd0,2)/boot/grub root=hd0,2 sh:grub> set menu_color_normal=white/blue sh:grub></pre>
<code>timeout</code>	在GRUB 2中， <code>timeout</code> 只是一个变量，用于指定默认的引导等待时间，使用户有机会会选择引导不同的操作系统，如“ <code>set timeout=10</code> ”，表示默认的引导超时时间为10秒

此外，Linux系统内核还可以接受一定的命令行选项或引导参数，如`root=value`、`ro`、`rw`、`single`或`init=value`等，可在引导时由GRUB传递给系统内核，以便系统内核能够正确地确定硬件配置等，解释如下：

- `root=value`——指定用做“/”文件系统的磁盘分区名。如果/`boot`目录位于单独的磁盘分区，即存在一个单独的/`boot`文件系统，`root`引导参数值应为/`boot`文件系统分区。因此，`root`参数指定的设备必须是一个可安装的文件系统分区，其中含有“/”（或/`boot`）文件系统。利用`root`引导参数，GRUB能够知道在引导期间采用哪一个设备或磁盘分区作为“/”文件系统。通常，`root`引导参数值是系统内核所在设备或磁盘分区，且在编译时就已事先确定。若想修改这一参数值，如使用系统盘的第3个分区引导Linux系统，可以使用“`root=(hd0,3)`”、“`root=/dev/sda3`”或“`root=UUID=...`”作为引导参数。其中，UUID是另外一种设备文件名的表达方式（参见/`etc/fstab`文件）。

- **ro或rw**——ro表示首先以只读方式安装“/”文件系统，使fsck能够检测处于静止状态的文件系统。rw表示以读写方式安装“/”文件系统（这是系统内核采用的默认安装选项）。
- **quiet**——表示仅需输出简明的引导信息，以便用户能够清晰地了解系统引导期间执行的动作。
- **splash**——表示在系统引导期间使用指定的徽标作为背景图像。
- **locale**——表示设置系统引导期间使用的语言环境。例如，“locale=zh_CN.UTF-8”表示把系统启动过程中的语言环境设置为简体中文。
- **single**——如果存在，single表示把系统引导至单用户的恢复模式（recovery mode）。
- **init**——设置系统内核完成系统引导过程之后调用的第一个进程。如果未设置或设置的程序文件不存在，系统内核将会按照/sbin/init、/etc/init、/bin/init和/bin/sh的顺序，尝试调用第一个发现的程序。如果所有的尝试均失败，系统只能转入瘫痪状态。

初始的grub.cfg配置文件通常包含两个Ubuntu Linux引导菜单项（其中一个为系统维护模式），一个memtest86+内存测试项，一个Windows引导菜单项（如果安装了双引导系统），以及一个空的custom引导菜单项。下面是安装Ubuntu 9.10 Linux系统之后生成的一个GRUB配置文件：

```
$ cat /boot/grub/grub.cfg
.....
### BEGIN /etc/grub.d/00_header ###
.....
set default="0"
.....
insmod ext2
set root=(hd0,2)
.....
if [ ${recordfail} = 1 ]; then
    set timeout=-1
else
    set timeout=10
fi
### END /etc/grub.d/00_header ###

### BEGIN /etc/grub.d/05_debian_theme ###
set menu_color_normal=white/black
set menu_color_highlight=black/white
### END /etc/grub.d/05_debian_theme ###

### BEGIN /etc/grub.d/10_linux ###
menuentry "Ubuntu, Linux 2.6.31-14-generic" {
    .....
    set quiet=1
    insmod ext2
    set root=(hd0,2)
    search --no-floppy --fs-uuid --set 693f43ef-d353-4a79-985f-bdd6929deca8
    linux /boot/vmlinuz-2.6.31-14-generic root=UUID=693f4... ro quiet splash
    initrd /boot/initrd.img-2.6.31-14-generic
}
menuentry "Ubuntu, Linux 2.6.31-14-generic (recovery mode)" {
    .....
```

```

        insmod ext2
        set root=(hd0,2)
        search --no-floppy --fs-uuid --set 693f43ef-d353-4a79-985f-bdd6929deca8
        linux /boot/vmlinuz-2.6.31-14-generic root=UUID=693f43ef-... ro single
        initrd /boot/initrd.img-2.6.31-14-generic
    }
    ### END /etc/grub.d/10_linux ###

    ### BEGIN /etc/grub.d/20_memtest86+ ###
    menuentry "Memory test (memtest86+)" {
        linux16 /boot/memtest86+.bin
    }
    menuentry "Memory test (memtest86+, serial console 115200)" {
        linux16 /boot/memtest86+.bin console=ttyS0,115200n8
    }
    ### END /etc/grub.d/20_memtest86+ ###

    ### BEGIN /etc/grub.d/30_os-prober ###
    menuentry "Microsoft Windows XP Home Edition (on /dev/sda1)" {
        insmod ntfs
        set root=(hd0,1)
        search --no-floppy --fs-uuid --set 4c54a5a254a58ef0
        drivemap -s (hd0) ${root}
        chainloader +1
    }
    ### END /etc/grub.d/30_os-prober ###

    ### BEGIN /etc/grub.d/40_custom ###
    # This file provides an easy way to add custom menu entries.  Simply type the
    # menu entries you want to add after this comment.  Be careful not to change
    # the 'exec tail' line above.
    ### END /etc/grub.d/40_custom ###
$

```

此外，/boot/grub 目录中的文件也不同于早期版本的传统GRUB。在GRUB 2中，这个目录包含GRUB引导程序本身使用的配置文件grub.cfg、初始引导映像文件（bootimg）、核心引导映像文件（coreimg）、设备映射文件（device.map）、各种帮助文件（如command.lst等）、一系列需要动态加载的引导模块文件（如ext2.mod），以及说明模块之间相互依赖关系的描述文件（如moddep.lst）等。如果其中的文件不全或有误，可以运行grub-install命令，重新创建或生成各种必要的文件。

15.1.4 GRUB实用程序

GRUB 2提供若干新的工具，如grub-install、grub-setup、grub-mkconfig、grub-set-default、grub-probe、grub-mkimage、grub-mkrescue、grub-mkdevice.map以及update-grub等。其中最常用的是grub-install和update-grub两个命令。下面摘其要点简述之。

1.grub-install命令

grub-install是GRUB 2实现系统引导的主要程序，用于检测系统引导过程需要的每一个组成部分，创建必要的目录（如\${root-dir}/boot/grub）；把*.mod、*.lst和*.img文件复制到\${root-dir}/boot/grub目录；调用grub-probe命令检测磁盘、磁盘分区及可引导的操作系统；调用grub-

mkimage命令,生成核心映像文件core.img;调用grub-setup命令,把引导程序(bootimg)与核心映像文件(core.img)写入磁盘的引导扇区(MBR)及紧随其后的磁盘扇区。grub-install命令的语法格式简写如下:

```
grub-install [options] device
```

其中,device可以是一个GRUB设备名(如hd0),也可以是一个Linux系统的设备文件名(如/dev/sda)。这个命令的部分选项及使用说明如表15-2所示。

表15-2 grub-install命令的常用选项

选项	GNU选项	描述
	<code>--root-directory=dir</code>	把GRUB(包括映像文件和可自动加载的模块等)安装到指定目录而非根目录下的boot/grub子目录中。当系统处于维护模式,需要重新安装GRUB时,这个选项是非常有用的
	<code>--grub-setup=file</code>	使用指定文件替代默认的grub-setup
	<code>--grub-mkimage=file</code>	使用指定文件替代默认的grub-mkimage
	<code>--grub-mkdevicemap=file</code>	使用指定文件替代默认的grub-mkdevicemap
	<code>--grub-probe=file</code>	使用指定文件替代默认的grub-probe
	<code>--recheck</code>	检测设备映射文件device.map是否存在
-h	<code>--help</code>	显示命令的简单说明与用法

2.grub-setup命令

grub-setup命令的主要功能是把GRUB引导映像文件bootimg和core.img分别复制到引导设备的MBR扇区及紧随其后的磁盘扇区中。这个命令通常是由grub-install命令调用的,无需单独运行。grub-setup命令的语法格式简写如下:

```
grub-setup [options] device
```

其中,device可以是一个GRUB设备名,也可以是一个Linux系统的设备文件名。这个命令的部分选项及使用说明如表15-3所示。

表15-3 grub-setup命令的常用选项

选项	GNU选项	描述
-b file	<code>--boot-image=file</code>	使用指定的文件作为引导映像文件,其默认值是/boot/grub/bootimg
-c file	<code>--core-image=file</code>	使用指定的文件作为核心映像文件,其默认值是/boot/grub/coreimg
-d dir	<code>--directory=dir</code>	使用指定目录中的GRUB文件,默认的目录是/boot/grub
-m file	<code>--devicemap=file</code>	使用指定的文件作为设备映射文件,其默认值是/boot/grub/device.map
-h	<code>--help</code>	显示命令的简单说明与用法

3.grub-mkimage命令

grub-mkimage命令用于生成可引导的核心映像文件(即core.img)。核心映像文件通常会复制到紧随MBR之后的磁盘扇区中。grub-mkimage命令的语法格式简写如下:

```
grub-mkimage [options] [modules]
```

grub-mkimage命令的部分选项及使用说明如表15-4所示。

表15-4 grub-mkimage命令的常用选项

选项	GNU选项	描述
-d dir	--directory=dir	使用指定目录中的映像文件和模块文件。在Intel x86 系列机中，默认的目录是 /usr/lib/grub/i386-pc
-m file	--memdisk=file	用于指定core.img文件中应包含的tar 档案文件，如grub.cfg等小型文件
-p dir	--prefix=dir	用于指定GRUB目录前缀，默认的目录是/boot/grub
-o file	--output=file	把grub-mkimage命令的输出写入指定的映像文件，默认值是标准输出
h	-help	显示命令的简单说明与用法

4.grub-mkconfig命令

grub-mkconfig命令主要用于生成grub.cfg配置文件，其语法格式简写如下：

```
grub-mkconfig [options]
```

grub-mkconfig命令的部分选项及使用说明如表15-5所示。

表15-5 grub-mkconfig命令的常用选项

选项	GNU选项	描述
-o file	--output=file	把grub-mkconfig命令的输出写入指定的配置文件，默认值是标准输出
h	-help	显示命令的简单说明与用法

例如，如果grub.cfg文件不完整，尤其是其中没有包含Windows系统引导菜单项时，可以利用下列命令重新生成grub.cfg文件：

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
[sudo] password for gqxing:
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-2.6.31-14-generic
Found initrd image: /boot/initrd.img-2.6.31-14-generic
Found memtest86+ image: /boot/memtest86+.bin
Found Microsoft Windows XP Home Edition on /dev/sda1
done
$
```

5.grub-mkrescue命令

grub-mkrescue命令调用grub-install命令，生成一个简单的GRUB维护映像文件，其语法格式简写如下：

```
grub-mkrescue [options] image
```

其中，image是一个文件名，用于存储生成的引导映像。这个命令的部分选项及使用说明如表15-6所示。

表15-6 grub-mkrescue命令的常用选项

选项	GNU选项	描述
	<code>--image-type=type</code>	用于指定生成的引导映像的类型，如floppy或cdrom，默认值是cdrom，即生成一个iso映像文件
	<code>--pkglibdir=dir</code>	采用指定目录中的映像文件，而非默认目录（/usr/lib/grub/i386-pc）中的映像文件
	<code>--grub-mkimage=file</code>	使用指定文件替代默认的grub-mkimage
-h	-help	显示命令的简单说明与用法

例如，为了生成一个软盘或CD/DVD系统维护介质，可以分别使用下列命令：

```
$ sudo grub-mkrescue -image-type=floppy /tmp/grub-rescue.floppy
$ dd if=/tmp/grub-rescue.floppy of=/dev/fd0
$ sudo grub-mkrescue -image-type=cdrom /tmp/grub-rescue.iso
$ sudo wodim /tmp/grub-rescue.iso
$
```

6.update-grub命令

update-grub命令只是grub-mkconfig命令的一个特例，采用Shell脚本的形式运行一个固定的grub-mkconfig命令，即“grub-mkconfig -o /boot/grub/grub.cfg”命令。其主要功能是利用/etc/default/grub文件和/etc/grub.d目录中的脚本文件生成grub.cfg配置文件。

例如，可以直接利用下列简单命令重新生成grub.cfg文件：

```
$ sudo update-grub
[sudo] password for gqxing:
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-2.6.31-14-generic
Found initrd image: /boot/initrd.img-2.6.31-14-generic
Found memtest86+ image: /boot/memtest86+.bin
Found Microsoft Windows XP Home Edition on /dev/sda1
done
$
```

15.1.5 安装或修复GRUB

在安装Ubuntu Linux系统时，GRUB将会随之一同安装到系统盘的MBR及Linux系统的/boot目录或文件系统中。如果准备在一台计算机中同时安装Windows与Linux两个系统，通常需要首先安装Windows，然后再安装Linux系统，否则无法启动Linux系统。如果因故缺失，或在安装Linux系统之后又重新安装了Windows系统，致使部分GRUB映像被覆盖，则无法正常启动Linux系统。

在此情况下，可以参照第1章介绍的安装步骤，使用下载的Ubuntu Linux系统的ISO映像文件，或刻录的CD/DVD安装介质启动系统（在安装界面选择“试用Ubuntu而不改变计算机中的任何内容（T）”），把系统引导至基于内存的Linux系统运行环境，安装或修复GRUB，从而修复已安装到计算机磁盘中的Linux系统。

例如，为了重新安装或修复GRUB，首先选择“应用程序→附件→终端”菜单，进入一个终端窗口，输入下列命令，查询Ubuntu Linux系统分区：

```
$ sudo fdisk -l
[sudo] password for gqxing:

Disk /dev/sda: 40.0 GB, 40000536576 bytes
255 heads, 63 sectors/track, 4863 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x64cc19d0

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1  *           1         1824     14651248+   7   HPFS/NTFS
/dev/sda2             1825         4731     23350477+   83   Linux
/dev/sda3             4732         4862     1052257+    82   Linux swap / Solaris

$
```

然后使用下列命令，把Ubuntu Linux系统分区安装到某个临时目录：

```
$ sudo mount /dev/sda2 /mnt
$
```

如果/boot位于单独的磁盘分区（如/dev/sda3），还需要把/boot文件系统安装到/mnt目录中：

```
$ sudo mount /dev/sda3 /mnt/boot
$
```

使用下列命令，设定当前的虚拟根目录，进入Ubuntu Linux系统分区：

```
$ sudo chroot /mnt
$
```

必要时可以利用编辑器，修改/etc/default/grub文件，或/etc/grub.d目录中的脚本文件，然后运行下列命令，生成新的配置文件（如果直接修改/boot/grub/grub.cfg配置文件，注意不要运行update-grub命令）：

```
$ update-grub
[sudo] password for gqxing:
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-2.6.31-14-generic
Found initrd image: /boot/initrd.img-2.6.31-14-generic
Found memtest86+ image: /boot/memtest86+.bin
Found Microsoft Windows XP Home Edition on /dev/sda1
done
$
```

使用下列grub-install命令，把GRUB的映像文件安装到系统盘/dev/sda的MBR等磁盘扇区中：

```
$ sudo grub-install /dev/sda
[sudo] password for gqxing:
Installation finished. No error reported.
This is the contents of the device map /boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(hd0)    /dev/sda
$
```

使用Ctrl-D组合键或exit命令退出虚拟根目录

```
$ exit
$
```

使用下列unmount命令卸载/mnt文件系统（在第二种情况下，尚需事先卸载/mnt/boot文件系统）：

```
$ sudo umount /mnt/boot
$ sudo umount /mnt
$
```

一旦安装，GRUB将会自动成为系统默认的引导程序。最后，关闭终端窗口，从设定的磁盘中重新引导系统。

如果上述命令出错，或不知道如何确定引导设备，也可以查阅/boot/grub/device.map文件，其中定义了GRUB设备名与Linux设备文件名之间的映射关系：

```
$ cat /boot/grub/device.map
(hd0)    /dev/sda
$
```

如果在此修复过程中出错，导致系统无法正常引导，而进入GRUB Shell时，可以参照下列步骤，利用GRUB命令及备份的grub.cfg文件（事实上只需记住Ubuntu Linux系统安装在哪一个磁盘分区上以及系统内核的版本即可），启动Ubuntu Linux系统。然后再按照前述步骤，重新安装GRUB引导程序（在这个例子中，Linux系统安装在第二个磁盘分区）：

```
sh:grub> insmod ext2
sh:grub> set root=(hd0,2)
sh:grub> linux /boot/vmlinuz-2.6.31-14-generic root=/dev/sda2 ro quiet splash
[Linux-bzImage, setup=0x3400, size=0x3b26e0]
sh:grub> initrd /boot/initrd.img-2.6.31-14-generic
[Initrd, addr=0x2f21e000, size=0x749e31]
sh:grub> boot
```

15.2 初始引导过程概述

Linux系统的一个重要功能特性是采用一种开放的、用户能够控制的方法引导或启动操作系统。用户可以根据自己的需要，配置操作系统的引导过程：指定不同的引导程序、引导参数以及其他设置。

GRUB是一个较大的程序，由于MBR容量的限制，GRUB 2引导程序通常拆分为3个不同的程序，即boot.img、core.img及一个选用的文件系统引导模块（如ext2.mod）。文件系统引导模块是一组引导程序集合的总称，针对不同的Linux系统及其文件系统，GRUB 2提供一系列文件系统引导模块，如ext2.mod、ntfs.mod、jfs.mod以及xfs.mod等，可以在GRUB运行期间动态加载。

boot.img是最基本的初始引导程序，可用于引导GRUB，从而引导目标操作系统。通常，随着操作系统的安装，初始引导程序boot.img将会复制到系统盘的0号扇区（MBR）中。由于计算机的0号引导扇区只有512个字节的存储空间，这个映像文件恰好也是512个字节。在完成最

初的系统引导之后，初始引导程序将会加载并执行系统盘紧随MBR之后的62个连续扇区（1～62号扇区）中的核心引导程序（core.img）。

core.img是GRUB的核心引导程序映像，其主要功能（或目的）是显示系统引导菜单，在用户做出选择之后，根据grub.cfg文件的设置，加载并执行文件系统引导模块（如ext2.mod）。注意，core.img通常无法识别和理解任何文件系统，因而只能采用动态加载的方式，调用特定文件系统的引导程序模块。

文件系统引导模块仅支持特定的一种文件系统，例如，ext2.mod文件系统引导模块能够识别Ext2/3/4文件，可以加载并执行位于“/”或/boot文件系统中的系统内核。事实上，ext2.mod文件系统引导模块是GRUB引导程序与系统内核之间的一个桥梁。

下面以Intel x86系列计算机上安装的Ubuntu 9.10 Linux系统为例，概述Linux系统的基本引导步骤，如图15-5所示。

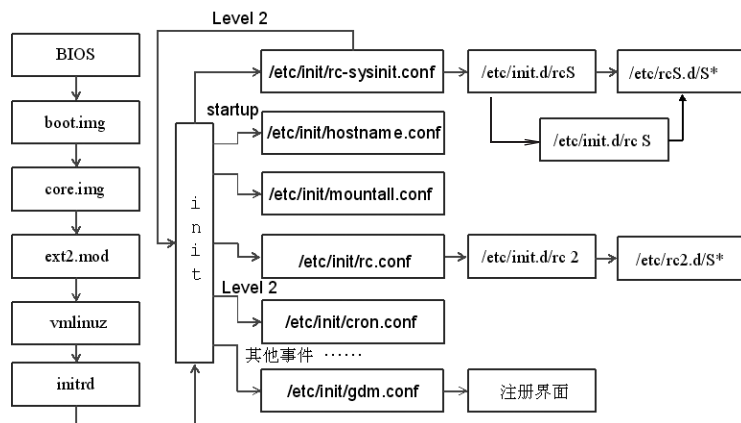


图15-5 Ubuntu Linux系统引导过程示意。

（1）加电自检。在加电或重新启动计算机时，计算机自动跳转并执行ROM中的BIOS引导程序，开始检测系统的硬件配置，运行硬件诊断程序，执行加电自检。然后找出能够引导系统的存储设备，如系统盘、软盘、CD/DVD、网络设备或USB移动盘等。根据0号扇区最后两个字节是否为0xAA55，确定其是否为MBR。一旦找出引导设备，BIOS将会立即加载0号扇区MBR中的初始引导程序（boot.img），最后把系统引导的控制权交由初始引导程序，执行最初的系统引导。

（2）执行初始引导程序（boot.img），进行初始引导，最后加载并启动系统盘等引导设备1～62号扇区中的核心引导程序（即core.img）。

（3）执行核心引导程序，读取grub.cfg文件，根据其中的设置，在控制台屏幕上显示一个操作系统引导菜单。

（4）根据用户的选择（使用上下箭头键，选择想要引导的操作系统，然后按下Enter键。在一定的时间内，如果用户一直置之不理，即没有做出任何选择，则引导事先设定的默认操作系统），及grub.cfg文件的设置，加载并执行ext2.mod文件系统模块，以便能够识别Ext2/3/4类型的文件系统，从而访问“/”或/boot文件系统分区。

（5）加载、执行系统内核文件/boot/vmlinuz-version-generic，加载、解压初始磁盘内存映像文件/boot/initrd.img-version-generic。根据引导参数，以只读或读写方式，把磁盘内存映像文

件安装成一个临时的虚拟“/”文件系统。同时，触发startup事件。

(6) 系统内核开始检测、设置Linux系统的硬件运行环境，配置CPU、内存、磁盘设备、I/O设备及其他硬件设备等，初始化软件模块，组织、初始化各种内部数据表，构造运行环境。然后再根据需要，加载其他必要的驱动程序模块。

(7) 启动/sbin/init程序，把引导过程的控制权转交给init进程，由init进程完成后续的系统生成过程。

(8) init进程检索/etc/init目录，监听各种系统事件。在收到startup事件之后，立即调度运行/etc/init/rc-sysinit.conf、/etc/init/hostname.conf和/etc/init/mountall.conf等作业。其中，前者用于确定系统的运行级（默认运行级为2），运行/etc/init.d/rcS脚本，进而以运行级S作为参数，运行/etc/init.d/rc脚本，按照优先顺序，依次启动/etc/rcS.d目录中的Shell脚本，配置系统控制台，最后运行telinit程序，触发“level 2”事件。后两个作业用于设置系统的主机名，检测与安装文件系统，触发各种文件系统事件（如filesystem、local-filefilesystems和all-filefilesystems等）。

(9) 根据/etc/init/rc-sysinit.conf作业触发的运行级事件，调度运行/etc/init/rc.conf作业，进而以运行级2作为参数，运行/etc/init.d/rc脚本，按照优先顺序，依次启动/etc/rc2.d目录中的Shell脚本，启动DNS、sshd、mysql、NFS、samba及apache2等服务进程。

(10) 根据启动过程中及部分作业（如mountall.conf）触发的事件，调度运行/etc/init目录中的作业，设置系统时钟、内核参数及网络连接，启动系统日志守护进程，启动cron、atd及anacron等守护进程，启动X服务器及GNOME显示管理器GDM等，直至控制台上显示出注册界面。至此，说明系统已经就绪，用户可以注册、访问Linux系统了。



在init进程开始运行之后，除了首先运行/etc/init/rc-sysinit.conf作业之外，/etc/init目录中的作业基本上是并发执行的，只要相应的事件已经触发。这也是Ubuntu 9.10 Linux系统启动过程比较快的主要原因。因此，上述步骤并不表明/etc/init目录中每个作业（如cron.conf、dbus.conf和网络-interface.conf等）的启动时间滞后于/etc/rcN.d目录中的启动脚本。即便是同时运行rc.conf作业，/etc/rcN目录中的启动脚本也是逐个运行的，故其通常滞后于/etc/init目录中的作业配置文件。

15.3 系统生成过程

系统启动过程的最后一个阶段是系统生成。在init守护进程开始运行时，大量的系统进程均未启动，需由init守护进程一一调度运行。

在系统生成阶段中，init守护进程主要根据/etc/init目录中的作业配置文件及系统或部分作业生成的事件，调度运行各种作业，从而启动各种服务进程，提供附加的系统与网络功能，最终完成系统启动过程中的后续处理任务。

init守护进程是系统引导后创建的第一个真正意义上的进程，是其他所有进程的父进程，除了少数几个内核进程，系统中的所有进程几乎都是init守护进程的直接或间接子进程。开始运行之后，init进程首先会检查/etc/init目录，读取其中的作业配置文件，利用套接字（或D-BUS总线）监听各种系统事件，调度运行相关的作业。

15.3.1 作业配置文件

与传统init进程基于运行级与/etc/inittab文件调度进程的方式不同，从6.10版开始，Ubuntu Linux系统开始采用Upstart init替代传统的init进程，以基于事件触发机制的方式调度进程，负责守护进程的管理与维护。但为保持兼容性，Upstart仍保留传统init的部分特性，且仍把Upstart init称做init进程。

init进程根据作业配置文件调度作业，所有的作业配置文件均位于/etc/init目录，其中定义了init进程应当调度运行的作业。init进程相当于一个监控程序，在启动之后，init进程将会检索/etc/init目录，读取其中定义的所有作业配置文件，监听系统中生成的任何事件，调度运行与事件相关的作业，从而启动相应的进程，跟踪、维护作业的状态，或终止作业的执行等。例如，等待文件系统安装的作业会由于安装事件的出现而启动，由于卸载文件系统事件的出现而终止。

1. 作业、任务与服务

/etc/init目录（或其子目录）中的文件是由init进程维护的作业配置文件，作业配置文件的主要用途是定义各种需由init进程调度运行的作业，每个作业配置文件用于定义一个作业。根据作业调度的方式，作业可以分为任务和服务。每个作业配置文件必须加“.conf”文件名后缀。

在作业的生命周期中，每个作业可以启动一个或多个不同的进程。其中，以exec或script关键字（两者只能使用其一）定义的进程称做主进程，在关键字exec之后，或在关键字script和end script之间可以指定任何命令或Shell脚本。当作业开始运行时，意味着启动exec或script关键字定义的命令或Shell脚本。一旦命令或Shell脚本终止运行，作业也就随之终止。

在/etc/init目录的作业配置文件中，每个作业通常都至少包含一个事件和一个命令（或Shell脚本），当定义的事件发生时，init进程就会调度运行相应的命令（或Shell脚本）。利用“initctl start”和“initctl stop”等命令，也可以直接运行或终止一个作业。

一个作业配置文件主要由五部分配置语句组成：作业描述、事件定义、进程定义，以及选用的环境设置与辅助设置等配置语句。其中，作业描述主要包括description关键字定义的简单作业描述，以及author关键字定义的作者及其联系方式等。事件定义主要包括“start on events（表示当指定的事件出现时应执行当前的作业）”和“stop on events（表示当指定的事件出现时应终止当前的作业）”等配置语句。进程定义主要包括exec和script关键字定义的命令或Shell脚本。有关作业配置文件的详细说明，可以查询init(5)手册页。

任务通常是由二进制目标代码或Shell脚本组成的，用于执行其内定的处理动作，完成后返回等待状态。实际上，任务是/etc/rcN.d目录中大多数Shell启动脚本的模拟实现。

在作业配置文件中，如果在作业指定的命令或Shell脚本之前增加一行仅包含关键字task的定义，表示当前的作业是一个任务，这意味着init进程必须确保当前作业的进程已开始运行，直至正常结束，终止运行之后也无需重启进程。例如，hostname.conf、hwclock.conf和mountall.conf等作业都是以任务方式实现的。示例如下：

```
$ cat /etc/init/hostname.conf
.....
description      "set system hostname"

start on startup

task
```

```
exec hostname -b -F /etc/hostname
$
```

第二种作业是服务，服务是一种需要持续运行的作业。例如，`cron`、`dbus-daemon`和`syslogd`等守护进程都是以服务方式实现的。`init`进程负责监控每一个服务，如果服务运行失败，`init`进程需要负责重启服务。如果系统管理员要求，或因某个事件要求停止运行一个服务，`init`进程也负责终止相应的服务进程。

如果没有明确指定，一个作业通常就是一个服务，这意味着，当启动一个作业时，只要进程已经处于运行状态，当前的作业就算完成。但是，进程一旦终止运行，即使其出口状态为0，也意味着需要重新调度运行指定的服务。示例如下：

```
$ cat /etc/init/network-interface.conf
.....
description      "configure network device"

start on net-device-added
stop on net-device-removed INTERFACE=$INTERFACE

instance $INTERFACE

pre-start script
    mkdir -p /var/run/network
    exec ifup --allow auto $INTERFACE
end script

post-stop exec ifdown --allow auto $INTERFACE
$
```

此外，还可以利用`respawn`配置语句，不管作业是正常终止还是异常终止，确保`init`进程能够自动地重新启动相应的进程。

为了限制一个作业重启的次数或重启的时间间隔，可以使用下列配置语句：

```
respawn limit count interval
```

其中，`count`表示作业重新调度运行的次数限制。`interval`表示作业重新启动的时间间隔。当一个作业在指定的时间间隔内重启的次数达到指定的限制，可以认为作业可能存在问题，因而无需再次重新启动。

2. 事件定义

事件是一种状态或状态变化信息，这种状态或状态变化信息可以来自系统内部，也可以来自系统外部，可以是系统预定义的，也可以是应用程序自定义的。例如，引导程序运行期间触发的`startup`事件，系统进入运行级2时触发的“`runlevel 2`”事件，以及插入移动存储介质触发的文件系统安装事件等。此外，在作业配置文件中增加“`emits event`”语句，也可以生成事件，利用“`initctl emit`”命令，还可以随时触发任何事件。目前，`init`进程能够识别的事件可以分为下列三种类型。

- 边界事件（Edge event）——是一个描述系统状态分界点的字符串，如`startup`、`starting`、`started`、`stopping`和`stopped`等。一旦定义的事件出现，系统将会调度运行或终止等待事件发生的作业。例如，“`start on startup`”表示在启动系统时执行相应的作业。
- 运行级事件（Level event）——类似于边界事件，其差别是除了描述性的字符串之外，

还有一个相关的值，如up或down等。字符串值的变动将会触发一个运行级事件和一个同名的边界事件。例如，“start on runlevel 2”表示当系统进入运行级2时执行相应的作业。

- 时间点事件（Temporalevent）——用于执行诸如“15分钟之后启动”、“每日”或“两点半”执行之类的动作。

关键字start on用于定义一个或一组事件，表示当指定的事件出现时，需要自动调度运行当前的作业，执行当前作业中定义的进程。其语法格式简写如下：

```
start on events [[key=]value].....
```

当需要同时指定多个事件时，可以采用and和or运算符，表示事件的逻辑依赖关系。如果逻辑表达式比较复杂，可以在前后增加一对圆括号（因而在必要时可以跨行）。除了定义的事件之外，还可以采用“key=value”的形式，匹配指定环境变量的预期值。如果事件提供的环境变量的顺序是已知的，也可以省略环境变量的名字。此外，指定变量值时还可以使用通配符。例如：

```
start on started gdm or started kdm
start on device-added SUBSYSTEM=tty DEVPATH=ttyS*
```

类似的，当采用关键字start on定义的一个或一组事件出现时，需要自动终止当终止当前的作业。其语法格式简写如下：

```
stop on events [[key=]value].....
```

在作业配置文件中，可以采用下列语句，在作业的运行过程中生成各种事件，包括状态变化事件等：

```
emits events
```

此外，系统管理员还可以采用initctl命令与init进程进行通信。实际上，利用initctl命令也能够启动、终止和查询作业，触发指定的事件等。例如，利用下列initctl命令，也可以直接生成任何事件：

```
initctl emit events [key=value].....
```

利用下列initctl命令，还能够列出当前的作业及其运行状态：

```
$ initctl list
avahi-daemon start/running, process 669
mountall-net stop/waiting
rc stop/waiting
rsyslog start/running, process 566
tty4 start/running, process 781
udev start/running, process 403
.....
$
```

此外，利用“initctl help”命令，可以查询initctl命令支持的每一个子命令。如果使用下列命令，还可以给出任何子命令的更多说明信息：

```
$ initctl [list | start | stop | status] --help
```

3. 进程定义

目前, Ubuntu Linux系统主要采用exec与script两种配置语句定义作业。其中, exec表示启动随后给出的单个命令, exec的语法格式如下:

```
exec command arguments
```

例如, 下面的内容取自/etc/init/hostname.conf作业配置文件, 表示当出现startup事件时执行指定的hostname命令, 设置系统的主机名:

```
$ cat /etc/init/hostname.conf
.....
description      "set system hostname"

start on startup

task
exec hostname -b -F /etc/hostname
$
```

script配置语句用于定义一组需要执行的命令, 这相当于执行一个内置的Shell脚本文件, 其语法格式如下:

```
script
command-list
end script
```

例如, 下面是取自/etc/init/dmesg.conf作业配置文件的一个片段

```
$ cat /etc/init/dmesg.conf
.....
start on runlevel [2345]

task
script
    savelog -q -p -c 5 /var/log/dmesg
    dmesg -s 524288 > /var/log/dmesg
    chgrp adm /var/log/dmesg
end script
$
```

从上述例子中可以看出, exec与script的功能基本上是一样的, 只不过前者执行的是单个命令, 后者执行的是一组命令。

除了主进程之外, 一个作业配置文件还可以定义4个附加的进程。例如, 如果在exec或script关键字前面冠以pre-start或post-start, 还可以定义附加的进程, 以便在主进程运行之前或之后执行指定的命令或Shell脚本。同样, 如果在exec或script关键字前面冠以pre-stop或post-stop, 也可以定义附加的进程, 以便在主进程终止运行之前或之后执行指定的命令或Shell脚本。

4. 运行级仿真

传统的init进程利用运行级及运行级的变化确定何时以及怎样启动或终止进程。而Ubuntu Linux系统中的init进程不采用运行级的概念。为了便于从基于运行级的系统迁移到基于事件的系统, 同时兼顾其他软件的兼容性, Ubuntu Linux系统采用运行级仿真运行方式。

运行级实际上是系统定义的一种运行模式或运行状态, 故运行级有时也称做运行状态(Run

State)。在不同的运行级中，系统能够提供不同的功能或服务。从系统管理的角度来看，运行级实际上是一种软件配置方式，在任何一个运行级中，只有一组选定的守护进程才能运行。

采用运行级定义系统的运行模式时，针对每一个运行级，早先（也包括现在）的Linux系统中都定义了一组选定的守护进程，以便构建一种特定的运行环境，从而使系统能够满足不同的工作需求。Linux系统可以处于不同的运行状态，但在任何时刻，Linux系统只能处于8种可能的运行级之一。例如，常见的多用户工作模式，或用于系统维护目的的单用户工作模式等。

定义运行级的目的是组织和划分系统进程的适用场合，根据不同的工作模式和功能需求确定不同的进程配置。因此，每个运行级具有不同的作用，以适应不同的功能需求，守护进程的主要作用是激活硬件设备，提供网络服务，以及维护系统日志等。每个系统进程既可属于一个特定的运行级，也可用于多个运行级。这就需要使用运行级划分系统的工作模式，相应地配置一组选定的系统进程。每个进程只能在其定义的运行级中才能运行。

在Linux系统中，init进程可以识别下列7个主要运行级（0~6）和一个内部定义的单用户运行级，即S（或s）运行级。

- 0——关机。用于停止Linux操作系统。
- 1——单用户工作模式。主要用于系统维护。
- 2——多用户运行模式。这是Ubuntu Linux系统默认的运行级，在完成系统初始化过程之后，Linux系统将会自动进入这个运行级。
- 3、4、5——暂未定义，可供用户定义其他多用户工作模式。
- 6——重新启动系统。这个运行级包括两个工作过程：首先是停止Linux系统，其处理步骤和内容与运行级0完全一样；然后是重新引导Linux系统，其效果如同加电启动系统。在进行系统配置、调整系统参数，以及安装系统软件包或更新Linux系统时，通常需要用这一运行级。
- S——系统内部定义的单用户恢复模式。在系统引导过程中，如果选用恢复模式，系统将会自动进入这一运行级。系统管理员可以通过控制台上访问系统，执行系统维护任务。

若想改变系统的运行级，超级用户可以运行init或telinit命令。如需了解系统当前所处的运行级，可以使用who或runlevel命令。例如：

```
$ who -r
run-level 2 2009-11-23 20:47
$ runlevel
N 2
$
```

在上述两个命令的输出信息中，“run-level 2”表示当前的系统运行级为2，即多用户图形界面运行模式。“2009-11-23 20:47”表示最近一次改变系统运行级的日期和时间。N表示先前的运行级不确定，2表示当前的系统运行级。

/etc/init/rcN文件中定义的rcN作业运行/etc/init.d/rc脚本。通过/etc/rcNd目录中的符号链接文件，这个脚本又调度运行/etc/init.d目录中的Shell启动脚本，仿真传统init进程的传统做法。当系统进入某个运行级时，Ubuntu Linux系统利用运行级作为参数，使用rcN作业启动传统的Shell脚本。当系统离开当前的运行级时，则不采取任何动作。Ubuntu Linux系统也实现了runlevel与telinit等实用程序，保持与System V系统的兼容性。

15.3.2 rc-sysinit.conf作业

在系统内核安装虚拟的“/”文件系统之后将会生成一个startup事件,这也是系统生成的第一个事件,这个事件将会触发init进程考察/etc/init/rc-sysinit.conf、hostname.conf和mountall.conf等配置文件,调度运行相应的作业。其中,rc-sysinit.conf是其中最重要的一个作业配置文件,用于实现传统的系统生成任务。

rc-sysinit.conf作业在设置部分运行级变量之后,首先检索/etc/inittab文件是否存在,检查系统内核的引导参数,确定系统的默认运行级。如果/etc/inittab文件存在,利用其中的initdefault定义,确定默认的运行级。

如果/etc/inittab文件不存在,再按照系统内核的引导参数,确定系统最终需要进入的运行级:

- 如果系统内核的引导参数为emergency, init进程将会执行/sbin/sulogin命令,进入系统维护模式。
- 如果系统内核的引导参数是一个表示运行级的数字或字符(如0~6、s或S),则使用运行级作为参数,运行telinit命令,从而触发运行级事件。
- 如果系统内核的引导参数是single,表示进入单用户运行模式。

在进入最终的运行级之前,首先执行/etc/init.d/rcS脚本,进而以S作为参数,调度运行/etc/init.d/rc脚本,依次执行/etc/rcS.d目录中以S为起始字符的Shell启动脚本。/etc/rcS.d目录中的Shell脚本用于执行部分系统初始化任务,如设置系统控制台与键盘,设置PCMCIA硬件支持等。

最后,利用先前确定的运行级(或默认运行级2)作为参数,调用telinit命令,生成一个运行级事件,触发init进程开始调度运行/etc/init/rc.conf作业配置文件。

/etc/init/rc-sysinit.conf文件的内容如下:

```
$ cat /etc/init/rc-sysinit.conf
.....
start on filesystem
stop on runlevel
.....
env DEFAULT_RUNLEVEL=2
.....
env RUNLEVEL=
env PREVLEVEL=

task

script
    # Check for default runlevel in /etc/inittab
    if [ -r /etc/inittab ]
    then
        eval "$(sed -nre 's/^[^#] [^:]*:([0-6sS]):initdefault:.*$/DEFAULT_RUNLEVEL
        ="\1"/p' /etc/inittab || true)"
    fi

    # Check kernel command-line for typical arguments
    for ARG in $(cat /proc/cmdline)
    do
        case "${ARG}" in
```



```
-b|emergency)
    # Emergency shell
    [ -n "${FROM_SINGLE_USER_MODE}" ] || sulogin
    ;;
[0123456sS])
    # Override runlevel
    DEFAULT_RUNLEVEL="${ARG}"
    ;;
-s|single)
    # Single user mode
    [ -n "${FROM_SINGLE_USER_MODE}" ] || DEFAULT_RUNLEVEL=S
    ;;
esac
done

# Run the system initialisation scripts
[ -n "${FROM_SINGLE_USER_MODE}" ] || /etc/init.d/rcS

# Switch into the default runlevel
telinit "${DEFAULT_RUNLEVEL}"
end script
$
```

15.3.3 init进程与/etc/init目录

理论上，可以在/etc/init和/etc/rcN.d目录中定义系统生成过程中需要调度运行的任何作业或启动脚本。但在实际应用过程中，这些目录中定义的作业或创建的启动文件大体上可以分为三类：一类是在系统进入指定运行级之前尚需执行的部分初始化任务；二是在进入指定的运行级之后需要启动的各种守护进程；三是启动应用程序提供的守护进程。

在/etc/init目录中，除了rc-sysinit.conf之外，还有hostname.conf和mountall.conf等作业配置文件也依赖于startup事件。因此，当startup事件出现时，init进程还会调度运行hostname.conf作业，设置系统的主机名，运行mountall.conf作业，根据/etc/fstab文件的要求，检测、修复与安装文件系统。

在运行过程中，作业也可以生成各种事件，从而引起init进程调度运行其他作业，如表15-7所示。例如，mountall.conf作业能够生成local-fileSystems、filesystem及all-fileSystems等文件系统事件，其中的local-fileSystems事件会引起init进程调度运行dbus.conf作业，而filesystem和started dbus事件又会引起init进程调度运行avahi-daemon.conf作业，local-fileSystems和started dbus事件引起NetworkManager.conf作业开始运行，最终导致/etc/init目录中的作业能够按照事件的依赖关系开始运行，以完成系统的生成。

表15-7 作业配置文件简单说明

作业配置文件	简单说明
avahi-daemon.conf	用于启动Avahi mDNS/DNS-SD守护进程，提供名字解析功能，实现网络的自动配置
dbus.conf	用于启动D-BUS系统消息总线的守护进程dbus-daemon。D-BUS是一个系统范围的IPC（Inter Process Communication）消息总线，用于系统守护进程与应用程序相互通信

(续表)

作业配置文件	简单说明
gdm.conf	用于管理X服务器, 启动GNOME显示管理器, 在控制台终端显示GNOME图形注册界面, 使用户能够注册访问系统
halconf	用于启动hald守护进程, 收集与维护系统硬件信息, 提供一个逻辑抽象层, 使应用程序能够访问硬件信息
hostname.conf	用于设置系统的主机名
hwclock.conf	根据硬件时钟调整系统时钟
idmapd.conf	用于启动rpc.idmapd守护进程, 为NFS服务器和NFS客户系统提供用户ID与名字映射服务
module-init-tools.conf	用于加载/etc/modules文件中指定的系统内核模块
mountall.conf	用于检测、修复与安装文件系统, 生成local-file-systems、file-system和all-file-systems等事件
networking.conf	用于配置虚拟网络设备
network-interface.conf	用于配置、激活网络接口, 设置IP地址等网络参数
network-manager.conf	用于选择并自动切换可用的网络连接。在同时支持多种网络连接手段(如无线网络与以太网)的计算机中实现网络的自动检测与切换
portmap.conf	用于启动portmap守护进程, 用于提供RPC程序号与TCP/UDP端口号转换等底层支持
procps.conf	用于读取/etc/sysctl.conf文件, 设置系统内核参数
rsyslog.conf	用于启动系统日志守护进程rsyslogd, rsyslogd是一个系统范围的日志守护进程, 其他许多守护进程均利用rsyslogd, 把自己的输出信息记录到系统日志文件中
rsyslog-kmsg.conf	用于实现/proc/kmsg内核消息到rsyslog系统日志的数据转储
sreadahead.conf	用于启动sreadahead守护进程, 以便采用预读方式, 把系统启动过程中需要运行的程序和数据文件事先读入内存缓存, 以提高启动速度, 加快系统的启动过程。注意, 仅当系统内存大于384MB时才有效

而且, 作为多用户工作模式的一部分, 当rc-sysinit.conf作业运行telinit命令, 触发一个运行级事件(level 2)时, init进程除了调度运行rc.conf作业外, 还会调度运行/etc/init目录中依赖于运行级事件的其他作业, 如表15-8所示。

表15-8 依赖于运行级事件的作业

作业配置文件	简单说明
acpid.conf	用于监听系统内核的ACPI事件, 管理和控制电源。对于笔记本电脑和部分台式机, 建议启用这个守护进程, 服务器需视具体情况酌情处理
anacron.conf	用于调度运行因系统停机等原因而错失执行机会的cron作业。anacron是对cron的补充。如果停机期间存在应当运行而没有运行的cron作业, 可由anacron辅助调度运行
apport.conf	用于获取系统内存映像, 自动生成系统瘫痪报告
atd.conf	用于启动atd守护进程, 调度运行利用at命令提交的定时执行作业; 在系统的平均负载较低时, 调度运行利用batch命令提交的作业

（续表）

作业配置文件	简单说明
cr on.conf	用于启动后台作业调度守护进程cr on。cr on是一个标准的Linux守护进程，用于周期性地调度定时执行的后台作业。系统应当总是启用cr on守护进程，atd或anacron则不一定需要
dmesg.conf	用于考察或控制系统内核环形缓冲区
rc.conf	用于调度运行传统的Shell启动脚本
ttyNconf	用于调度执行6个终端服务（其中N=1，2，…，6），确保当系统处于2和3等运行级时都能打开一个虚拟控制台

此外，在正常的系统启动过程中，通常不会处理control-alt-delete.conf作业配置文件。但在运行期间（包括系统生成过程中），如果用户在控制台上按下Ctrl-Alt-De1组合键，系统将会执行预定义的shutdown命令。例如：

```
$ cat /etc/init/control-alt-delete.conf
.....
description      "emergency keypress handling"
author           "Scott James Remnant <scott@netsplit.com>"

start on control-alt-delete

task
exec shutdown -r now "Control-Alt-Delete pressed"
$
```

15.3.4 init进程与/etc/rcN.d目录

当前，除了在/etc/init目录中创建作业配置文件，控制作业的运行状态之外，作为一种过渡阶段，/etc/init.d和/etc/rcN.d目录中仍然保留部分启动脚本文件。

1./etc/rcS.d目录

如前所述，/etc/init/rc-sysinit.conf作业在确定系统的运行级之后会运行/etc/init.d/rcS脚本，rcS脚本又会以运行级S作为参数，运行/etc/init.d/rc脚本，依次运行/etc/rcS.d目录中的脚本文件（参见表15-9），执行部分系统初始化任务。

表15-9 /etc/rcS.d目录中的部分Shell启动脚本

启动脚本	简单说明
S06keyboard	用于设置控制台键盘
S13pcmciautils	用于提供PCMCIA硬件支持
S35quota	根据/etc/fstab文件的要求，检测限额设置的一致性，启用限额控制机制
S49console-setup	用于设置控制台及其支持的字体和键盘映射等

2./etc/rc2.d目录

在运行/etc/init.d/rcS脚本之后，/etc/init/rc-sysinit.conf作业会利用最终确定的运行级作为参数，运行telinit命令，生成一个运行级事件。在Ubuntu Linux系统中，默认的运行级为2，因此

在“runlevel2”事件出现之后，init进程将会调度运行/etc/init/rc.conf作业，rc.conf又会以运行级2作为参数，运行/etc/init.d/rc脚本，检测/etc/rc2.d目录，启动/etc/rc2.d目录中以S为起始文件名的Shell脚本。/etc/init/rc.conf作业配置文件的内容如下：

```
$ cat /etc/init/rc.conf
.....
description      "System V runlevel compatibility"
author           "Scott James Remnant <scott@netsplit.com>"

start on runlevel [0123456]
stop on runlevel [!$RUNLEVEL]

export RUNLEVEL
export PREVLEVEL

task

exec /etc/init.d/rc $RUNLEVEL
$
```

/etc/init.d/rc脚本开始运行时，将会按照文件名的字符顺序，逐个运行/etc/rc2.d目录中以S为文件名起始字符的所有Shell脚本，依次启动相应的守护进程，从而完成最后的系统生成。

Ubuntu Linux系统的部分守护进程启动脚本仍然位于/etc/init.d目录，根据需及启动或终止的顺序，Linux系统采用符号链接的方式，把一组选定的Shell启动脚本分别链接到相应的/etc/rcN.d目录中，使之能够在进入不同的运行级时，执行一组特定的Shell脚本，而系统只需维护一套完整的Shell脚本。当需要在不同的运行级中运行同一个守护进程时，只要把守护进程的相应Shell脚本分别链接到多个/etc/rcN.d目录中即可。

在/etc/rcN.d目录中，每个符号链接文件的名称前部都有一个S或K字符，S表示在进入指定的运行级时启动相应的守护进程，K表示终止相应的守护进程。除了K或S之外，每个文件名还包含两个数字编号，表示启动或终止的顺序。通过改变数字的编号，可以改变启动或终止守护进程的顺序。数字编号越小，启动或终止时执行得越早。如果数字编号相同，则按整个文件名的字符排序原则确定先后次序。

/etc/rcN.d目录中的守护进程是由位于/etc/init.d目录中的rc脚本启动或关闭的。采用“/etc/rcN.d/script start”形式的命令，启动以S为起始文件名的守护进程。采用“/etc/rcN.d/script stop”形式的命令，终止以K为起始文件名的守护进程。其中，script是守护进程相应脚本的文件名。

表15-10给出了/etc/rc2.d目录中的部分Shell启动脚本，说明了在进入常规的多用户运行模式时都会启动哪些守护进程（或应用程序）。注意，在不同版本的Linux系统中，甚至在同一版本的Linux系统中，由于选择安装的软件包不同，/etc/rc2.d目录中的文件并不完全一样。实际上这一点并不重要，我们更关心的只是这样一种机制，以及怎样利用这种机制，把应用程序的启动（与关闭）脚本加到系统的生成过程中，使应用程序能够随着系统的启动而自动启动，但不关心目录中都有哪些启动文件，其功能如何。

在了解了作业配置文件和Shell启动脚本的基础上，可以根据系统的安装情况及具体配置，确定究竟应当启用和禁用哪些守护进程，从而达到提高系统性能的目的。

表15-10 /etc/rc2.d目录中的部分Shell启动脚本

启动脚本	简单说明
S15bind9	用于启动DNS域名服务器的named守护进程，实现域名与IP地址解析
S16ssh	用于启动OpenSSH守护进程sshd。sshd守护进程允许外部用户利用SSH协议访问本地系统。除非作为一个Linux系统，必须提供客户系统利用SSH协议访问本地系统的远程注册服务，否则不应启用这个守护进程。参见第18章“TCP/IP网络应用”
S19autofs	用于启动automount守护进程，以便在用户试图进入安装点，访问远程共享资源时，能够自动安装NFS网络文件系统
S19mysql	用于启动MySQL数据库服务器
S20nfs-kernel-server	用于启动NFS服务器的nfsd、nfsd4、lockd和rpc.mountd等守护进程，提供NFS服务器功能，公布/etc/exports文件中设定的目录或文件系统等共享资源
S20openbsd-inetd	用于启动inetd网络监控程序，调度执行传统的网络守护进程。inetd是一种特殊的网络总控服务，用于统一管理与调度运行多个网络守护进程。根据针对不同端口的网络访问请求，inetd将会启动相应的守护进程。例如，telnet通常利用端口23访问系统，如果检测到访问端口23来的telnet连接请求，inetd将会调度执行telnetd守护进程
S20samba	用于启动Samba服务器的守护进程smbd与nmbd，提供Linux与Windows系统之间的文件与打印资源共享服务。除非确实需要，否则不应启用这个守护进程
S20sysstat	用于启动系统活动数据收集器sadc，以便能够利用sar工具分析系统性能
S20vsftpd	用于启动FTP服务器，提供文件传输服务
S25bluetooth	用于启动蓝牙（Bluetooth）守护进程bluetoothd，管理蓝牙无线设备，如蓝牙无线键盘、鼠标和耳机等，支持基于蓝牙协议的拨号网络，以及连接以太网等。蓝牙是一种无线通信协议（并非802.11），支持便携式蓝牙无线设备，支持服务的发现、认证及人机接口设备等。有些笔记本电脑支持蓝牙无线设备，但大多数台式机、服务器或部分笔记本电脑并不支持蓝牙协议，因而应禁用这个守护进程
S50cups	用于启动通用UNIX打印系统（Common UNIX Printing System，CUPS）的守护进程cupsd
S50pulseaudio	用于启动PulseAudio音频服务器
S50rsync	用于启动rsync进程，实现远程文件复制
S91apache2	用于启动Apache服务器，提供网络浏览服务
S99rc.local	系统启动过程中最后执行的一个启动脚本。用户可以利用这个启动脚本，增加自己的初始化功能，从而避免自己编写一个完整的启动脚本

15.3.5 启动应用程序

1. 创建启动脚本

在启动脚本文件中，基本上都是采用case分支控制语句，按照start、stop和restart等情况，确定怎样启动、停止或重新启动相应的守护进程。因此，可以利用init进程和/etc/rcN.d目录的控制机制，创建自己的启动脚本，实现应用程序的自动启动。

2. 创建作业配置文件

除了创建启动脚本，也可以创建自己的作业配置文件，把需要执行的进程作为任务或服务加到/etc/init目录中，由init进程负责调度执行，实现应用程序的自动启动。

3. 修改rc.local文件

实际上还可以采用其他方法，在系统的引导过程中启动用户定义的应用程序。在系统引导过程中或在改变运行级时，init进程都会执行/etc/rc.local脚本。根据这一特点，用户可以在这个脚本文件的后面，增加自己的命令，执行必要的任务。这一做法比较简单，能够避免在/etc/init.d目录中自己编写启动脚本，然后在相应的/etc/rcN.d中建立符号链接文件。

15.4 login进程

15.4.1 login进程与passwd文件

当用户在注册界面或“login:”提示下输入用户名之后，Linux系统将会调用login进程，由login进程负责处理用户的注册。login进程首先会提示用户输入密码。在用户输入密码时，键入的字符通常不回显。读取密码之后，login进程将根据/etc/passwd文件，比较输入的注册用户名和密码。如果有一项不符，login进程将会再次提示用户输入用户名与密码，如果用户名和密码通过了验证，login进程将会按照/etc/passwd文件中“home-dir”与“login-shell”字段的设置（参见第9章“用户管理”），确定用户的初始工作目录和交互注册Shell，然后以用户名作为参数，通过exec系统调用，调度执行Shell程序。

15.4.2 Shell进程与profile文件

如上所述，用户注册后究竟调用哪一个Shell，取决于/etc/passwd文件中的定义（默认的注册Shell为/bin/bash）。当调用一个注册Shell时，Shell将会运行相关的启动文件，初始化各种必要的变量，设置运行环境。至于究竟运行哪些启动文件，取决于调用的是哪一个Shell。

这里以Bash为例，说明注册Shell的启动过程。在开始运行之后，Shell首先会读取并执行/etc/profile初始化文件，设置系统范围内的运行环境，如PS1等变量。期间，如果/etc/bash.bashrc文件存在，/etc/profile将会间接调用该文件，然后进入用户主目录，执行其中的profile（profile又调用.bashrc）等用户初始化文件，进一步定制自己的运行环境，最后进入GNOME桌面（当调用终端窗口时，取决于用户的身份，输出后缀为“\$”或“#”的命令提示符，开启Shell会话过程）。

至此，Linux系统的整个启动过程全部完成，系统已处于待命状态，用户可以访问GNOME桌面环境，打开终端窗口，输入任何Shell命令，访问Linux系统。

顺便提一下，当用户退出系统，也即退出Shell时，系统会自动执行~/bash_logout文件。因此，可以使用这个脚本文件执行某些善后处理，如清除临时文件等。

15.5 系统关机过程

15.5.1 使用shutdown命令关闭系统

shutdown命令采用一种比较安全的方式关闭系统。在真正开始关闭系统之前，系统将会不断地通知或提醒已经注册到系统中的所有用户，使用户能够早做准备，及时保存尚未完成的处

理工作，同时拒绝其他用户再注册到系统。shutdown命令也会向所有进程发送SIGTERM信号，使进程能够执行善后处理。最终，shutdown命令将会调用init进程，请求改变运行级。除了在指定的时间延迟之后实施实际的关机动作之外，shutdown命令也能够立即关机。shutdown命令的语法格式如下：

```
shutdown [-chkrHP] time [warning-message]
```

其中，“-r”选项用于重新引导系统。time用于指定停机的时间。时间参数可以采用不同的表示格式：如采用24小时制的hh:mm形式表示开始停机的绝对时间，其中hh表示小时，mm表示分钟；也可以采用相对时间形式+m，其中m表示开始停机前的缓冲时间（分钟数）。此外还可以使用英文单词now（也即+0），表示立即执行关机。warning-message用于指定发送给所有注册用户的消息。

在开始关机之前，系统通常会按照一定的时间间隔向注册的所有用户发出警告信息。下面的例子表示在输入shutdown命令之后，将会在凌晨1:10开始关机，中间按一定的时间间隔（每分钟一次）输出警告信息，关机后重新启动系统：

```
$ sudo shutdown -r 1:10

Broadcast message from gqxing@iscas
(/dev/pts/0) at 1:08 ...

The system is going down for reboot in 2 minutes!

Broadcast message from gqxing@iscas
(/dev/pts/0) at 1:09 ...

The system is going down for reboot IN ONE MINUTE!
```

15.5.2 使用init命令关闭系统

为了快速关机，可以直接使用“init 0”命令。在使用“init 0”命令关闭系统时，系统将会以0作为参数，执行/etc/init.d/rc脚本，然后执行/etc/rc0.d目录中以“Knn”为起始文件名的所有Shell脚本，最终关闭系统。示例如下：

```
$ sudo init 0
```

15.5.3 使用其他命令关机

除了shutdown和init命令之外，还可以使用halt、poweroff或reboot等命令关闭或重新启动系统。

第16章 作业调度与系统日志

在Linux系统中，cron、atd与anacron是三个重要的守护进程，负责调度各种定时运行的后台作业。本章主要讨论怎样实现后台作业调度，定时执行各种系统任务与用户程序，包括重复执行的任务以及一次性执行的作业，并给出相应的应用实例。同时简单介绍系统提供的日志功能与各种日志文件。

16.1 定时运行后台作业


在调度定时执行的系统任务与用户程序方面，cron、atd与anacron三个守护进程各有侧重与分工。cron是Linux系统中的主要作业调度程序，负责调度执行所有的系统服务；atd负责调度执行用户利用at（或batch）命令提交的定时作业；anacron是对cron的补充，如果存在停机期间应当运行而没有运行的cron作业，在系统启动之后，将由anacron协助调度执行。

利用后台作业调度机制，用户也可以提交自己的后台作业，以便能够定时、自动地执行自己的应用程序（包括系统命令和Shell脚本）。例如，在每天晚上的某一时间或每周的周末执行系统备份，按一定的时间间隔收集系统数据等。

表16-1给出了与作业调度有关的命令、守护进程、crontab文件、cron调度执行的Shell脚本，以及相关的控制文件。

表16-1 与作业调度有关的命令和文件

命令与守护进程	调度功能	crontab文件	调度的脚本文件	控制文件
crontab/cron	按照指定的时间重复执行指定的任务	/etc/crontab	/var/cronhourly/*	/etc/cron.deny（选用的）
		/etc/cron.d/*	/etc/cron.daily/*	/etc/cron.allow（选用的）
		/var/spool/cron/crontabs/*	/etc/cron.weekly/*	
			/etc/cron.monthly/*	
at/batch/atd	定时执行单个任务		/var/spool/cron/atjobs/*	/etc/at.deny（选用的） /etc/at.allow（选用的）
anacron	调度错失执行时间的作业	/etc/anacrontab	/etc/cron.daily/* /etc/cron.weekly/* /etc/cron.monthly/*	

 **注意**

crontab文件主要用于提交重复执行的任务，at命令则仅仅用于提交一次性执行的任务。

16.1.1 cron守护进程的调度过程

cron守护进程的主要功能是管理和调度以crontab文件形式提交的后台作业，按照指定的日期和时间自动执行指定的命令。在系统的启动过程中，通过执行/etc/init.d目录中的cron脚本，Linux系统将会自动启动cron守护进程。一经启动，cron守护进程将首先检测/var/spool/cron/

crontabs目录中是否存在用户定义的,并以用户名命名的crontab文件,然后检测/etc/crontab文件,最后检测/etc/cron.d目录中是否存在系统crontab文件。如果存在,cron守护进程将会执行下列任务:

- 读取并加载crontab文件中定义的任务,其中包括执行时间和相应的命令(或Shell脚本);
- 每分钟运行一次,检查已加载到内存中的每一项任务。如果某一任务的指定时间与当前的系统时间匹配,则调度执行相应的命令或Shell脚本;
- 在执行指定的命令或Shell脚本时,其中的任何输出信息将会通过电子邮件的形式发送给crontab文件的属主,或由crontab文件中MAILTO环境变量指定的用户;
- 检查/var/spool/cron/crontabs和/etc/cron.d目录,以及/etc/crontab文件自加载后其修改时间是否发生变化。如有变化,cron将会重新考察上述两个目录中的所有crontab文件,重新加载已经编辑或修改过的crontab文件。

为了利用cron守护进程,提交定时、重复执行的例行任务(命令或Shell脚本),可以选用/etc/crontab或/etc/cron.d目录中的一个crontab文件,手动编辑选定的文件,增加自己的定时执行任务。也可以使用crontab命令,在/var/spool/cron/crontabs目录中创建、编辑或删除自己的crontab文件(对于只需执行一次的命令或Shell脚本,可以使用at命令,提交atd守护进程,使之按照指定的时间调度运行)。

利用crontab文件,可以按时、按日、按周或按月调度执行各种日常的系统管理,使系统能够在指定的时间重复执行例行的维护任务,如根据预定的保留时间,轮换过时的日志文件,以及系统备份等。

此外,还可以利用crontab执行其他日常的系统管理与维护任务,定时备份数据库中的重要数据等,详见16.2节以及16.2.6小节中的应用实例。



请勿混淆crontab命令与所谓的crontab文件。实际上,Linux系统既提供一个crontab命令,用于创建、编辑、显示和删除/var/spool/cron/crontabs目录中的文件,也提供一个/etc/crontab文件。但通常所谓的crontab文件实际上只是一种通称,泛指/etc/crontab文件本身,以及/var/spool/cron/crontabs与/etc/cron.d等目录中的所有文件,尽管其文件名并非crontab。

除了人为因素,一经启动,cron守护进程通常会无休止地连续运行。而且,整个系统只能运行一个cron进程。这是因为系统使用/var/run/crond.pid文件作为封锁文件,防止用户执行多个cron进程。

16.1.2 at作业与atd守护进程

利用at命令,用户能够提交在指定的时间开始执行的作业。作业可以是一个命令,也可以是一个Shell脚本。类似于crontab,at命令允许用户调度执行一定的系统管理与维护任务。但与crontab文件不同的是,at命令提交的任务仅在指定的时间执行一次。执行之后,定期从其作业目录中删除相应的文件。因此,当需要一次性地定时运行某个命令或Shell脚本时,at命令是非常有用的。注意,在利用at命令提交作业时,应把命令或Shell脚本的输出信息重定向并存储到一个文件中,以备将来考察。

当用户利用at命令提交at作业时,系统将会在/var/spool/cron/atjobs目录中以文件方式保存用户提交的命令或Shell脚本,以及用户当前工作环境的副本。at作业文件采用较长的字符串命名,

文件名反映了提交的命令或Shell脚本在at作业队列中的位置, 以及执行状态。

文件名的第一个字符表示作业的类型和状态, 其中“a”表示利用at命令提交的作业, “b”表示利用batch命令提交的作业。如果文件名的第一个字符变为等号“=”, 说明作业已经执行完毕。文件名的第2~6个字符以十六进制的数值形式表示作业的队列号。

例如, a00008012eb8ea是一个利用at命令提交的作业, 作业的队列号为8。b0000b012f627f是利用batch命令提交的作业, 作业的队列号为11。文件名“=00003012eb8ea”意味着作业已经执行。

一经启动, atd守护进程即开始检测/var/spool/cron/atjobs目录中是否存在at作业, 同时随时监听用户是否利用at命令提交了新的作业。在完成at作业之后, atd守护进程将会重新命名/var/spool/cron/atjobs目录中的作业文件, 把文件名的第一个字符改为等号“=”, 文件名的其他部分保持不变。16.3节将会详细说明怎样提交定时执行的at作业。

此外, 还可以利用batch命令提交可在任何时间执行的命令或Shell脚本, 此时无需控制作业的执行时间。当存在一个或一组需要执行的命令, 但不关心究竟何时执行时, batch命令是特别方便的。atd守护进程将会对提交的at作业进行排队, 以便在系统负载允许的情况下调度执行。除非采用了重定向, 命令的标准输出和标准错误输出将会通过电子邮件的方式发送给用户。

16.1.3 调度错过执行时间的任务

anacron能够以指定的天数作为时间间隔, 周期地执行命令。与cron不同, anacron假定计算机系统并非不间断地运行。因此, 在一个非24小时连续运行的Linux系统中, 可以利用anacron协助cron控制每天、每周或每月调度执行的作业。anacron命令的语法格式简写如下:

```
anacron [-s] [-f] [-n] [-d] [-q] [-t newcrontab] [job] ...
anacron -u [-t anacrontab] [job] ...
```

其中, “-f”选项表示强制执行作业, 忽略相关的时间属性。“-u”选项表示仅仅更新作业的时间属性, 把作业的时间属性设置为当前日期, 但并不调度执行任何作业。“-s”选项表示依次执行每个作业。在前一个作业尚未结束运行之前, anacron通常不会开始调度执行新的作业。“-n”选项表示立即运行作业, 忽略/etc/anacrontab文件中规定的延迟时间。这个选项也蕴涵着“-s”选项。“-d”选项表示不创建后台进程。“-t”选项表示使用指定的文件替代默认的anacrontab文件。

当随着系统的启动开始运行时, anacron将会从默认的配置文件的/etc/anacrontab (除非另行指定) 中读取需由anacron调度执行的一系列作业定义。每个作业定义包含以天数为单位的时间周期, 以分钟为单位的时间延迟、作业标识符, 以及指定的命令或Shell脚本。

对于每个作业, anacron将会检查作业是否已经在最近的n天之内执行过, 其中的n表示指定的时间间隔。如果未执行, anacron将会在指定的时间延迟之后, 调度执行指定的命令或Shell脚本。

在指定的命令或Shell脚本执行结束之后, anacron将会在一个作业特定的时戳文件中记录作业的执行日期, 以便能够确定何时需要再次调度运行。anacron仅用这个日期进行时间计算。如果没有需要调度执行的作业, anacron将会结束运行。/var/spool/anacron目录用于存储anacron作业的时戳文件。

除非指定了“-d”选项, anacron将会创建一个后台子进程, 当子进程开始执行时立即结束

自己的运行。另外，除非指定了“-s”或“-n”选项，anacron还会在指定的延迟时间之后立即启动指定的作业。

如果作业经由标准输出或标准错误输出产生任何输出信息，这些输出信息将通过电子邮件的形式，送交运行anacron的用户（通常为root）。而一般的维护性信息则由cron送交rsyslogd处理。

anacron的配置文件/etc/anacrontab描述了anacron调度执行的作业。这个配置文件由两种文本行组成：作业描述行和环境变量赋值行。作业描述行的语法格式如下：

```
period delay job-identifier command
@period_name delay job-identifier command
```

其中，period是以天数为单位的时间间隔，delay是以分钟为单位的延迟时间。job-identifier可以包含任何非空格字符及斜杠字符“/”，用于标识anacron作业，同时用做作业时戳文件的名字。command可以是任何命令或Shell脚本。@period_name表示预定义的时间周期，当前唯一可取的一个有效值是@monthly，其作用是确保每月仅运行一次，而不管一个月究竟有几天。

环境变量赋值行的语法格式如下：

```
variable = value
```

其中，variable是环境变量名，value是变量的值。配置文件中允许有空行，或以注释符号“#”为起始字符的注释行。下面是Ubuntu Linux系统提供的默认anacrontab文件：

```
$ cat /etc/anacrontab
.....
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# These replace cron's entries
1 5    cron.daily nice run-parts --report /etc/cron.daily
7 10   cron.weekly nice run-parts --report /etc/cron.weekly
@monthly 15 cron.monthly nice run-parts --report /etc/cron.monthly
$
```

在上述配置文件中，run-parts是一个Shell脚本，用于执行指定目录参数中的所有Shell脚本文件（部分特殊的Shell脚本文件除外）。

16.2 调度重复执行的任务

这一节开始详细介绍怎样创建、编辑、显示和删除crontab文件，讨论如何实现定期、定时、自动执行系统的日常管理与例行维护任务，以及怎样控制对crontab文件的访问。

16.2.1 crontab文件及其工作原理

除了/etc/crontab文件之外，Ubuntu Linux系统提供的crontab文件通常均存储在/etc/cron.d目录中。此外，用户也可以创建以自己的注册名命名的crontab文件，以便调度运行自己的各种处理任务。用户创建的crontab文件均位于/var/spool/cron/crontabs目录中。

cron守护进程每分钟考察一次利用各种crontab文件定义的后台作业，并按照每个crontab文件中的指令，定时执行指定的命令或Shell脚本。crontab文件通常由若干指令组成，每个指令的

前5个字段用于设定日期和时间，最后一个字段用于设定应自动运行的命令或Shell脚本。这些信息告诉cron守护进程应在何时调度执行指定的命令或Shell脚本。

crontab文件分为系统crontab文件和用户crontab文件两种形式。其中，系统提供的crontab文件由下列两部分内容组成。第一部分是选用的，用于设置环境变量；第二部分包括一系列指令，每个指令由7个字段组成，中间以空格作为分隔符：

```
variable= value
minute hour day month week user command
```

用户定义的crontab文件由下列两部分内容组成。第一部分是选用的，用于设置环境变量；第二部分包括一系列指令，每个指令由6个字段组成，中间以空格作为分隔符：

```
variable= value
minute hour day month week command
```

运行时，cron将会自动设置若干环境变量，例如，把SHELL设置为/bin/sh，并根据crontab文件的属主，适当地设置LOGNAME和HOME变量。注意，在crontab文件中，用户可以强制修改HOME和SHELL变量的默认值，但不能强制修改LOGNAME变量。

除了LOGNAME、HOME和SHELL变量之外，cron还会寻找MAILTO变量，以便能够向用户报告运行结果。如果设置了MAILTO，cron将会向MAILTO变量定义的用户发送邮件。如果设置了MAILTO，但其变量值为空（MAILTO=""），则不会发送任何邮件。否则，cron将会向crontab文件的属主发送邮件。

在crontab文件的指令部分，每一行的前5个字段分别表示分、小时、日、月和周。如果是系统crontab文件，第6个字段是用户名，表示以什么身份运行后台作业。最后一个命令字段用于指定需要调度执行的命令或Shell脚本。根据前5个字段设定的日期和时间，系统能够自动地重复执行指定的命令或Shell脚本。注意，命令字段中可以同时指定多个命令，命令之间需加分号“;”分隔符。表16-2给出了前5个字段的说明。

表16-2 crontab文件中的时间字段

时间字段	取值范围
分	0~59
小时	0~23
日	1~31
月	1~12
周	0~7（0或7均表示星期日）

在crontab文件的每个时间字段中，还可以使用下列特殊字符，以提高时间定义的灵活性。

- 可以使用逗号并列多个数值，如“1,3,5,7,9”；
- 可以使用减号“-”指定一个数值范围，如“1-9”；
- 可以使用除号“/”和一个数字表示增量，如“1-9/2”，相当于“1,3,5,7,9”；
- 可以使用星号“*”通配符表示所有可能的合法数值；
- 除了数字之外，月和周两个字段还可以使用其英文名字的缩写（前3个字符），如feb表示二月，sun表示星期日等；

·可以在每一行的起始位置采用注释符号“#”增加一个注释行或加一个空行。

例如，下列crontab指令表示，每个星期五的下午4时，在系统控制台的终端窗口中显示一条提示信息：

```
0 16 * * 5 root echo -e "\n***** Timesheets Due *****" > /dev/pts/0
```

下面是Ubuntu Linux系统提供的/etc/crontab文件，其主要目的是按小时、日、周和月，分别调度运行位于cron.hourly、cron.daily、cron.weekly和cron.monthly目录中的Shell脚本，以便完成不同的系统维护任务。

```
$ cat /etc/crontab
.....
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || (cd / && run-parts --report /etc/cron.daily)
47 6 * * 7 root    test -x /usr/sbin/anacron || (cd / && run-parts --report /etc/cron.weekly)
52 6 1 * * root    test -x /usr/sbin/anacron || (cd / && run-parts --report /etc/cron.monthly)
$
```

第一个指令表示在每小时的17分，调度执行/etc/cron.hourly目录中的crontab文件；第2个指令表示在每天的6:25，调度执行/etc/cron.daily目录中的crontab文件；第3个指令表示在每个星期日的6:47，调度执行/etc/cron.weekly目录中的crontab文件；第4个指令表示在每月第一天的6:52，调度执行/etc/cron.monthly目录中的crontab文件。



注意

crontab文件中的每个指令必须由一个完整的逻辑文本行组成，即使文本行太长，需要占用多个物理行也是如此，中间不能有回车、换行字符。

16.2.2 创建和编辑crontab文件

在实际应用过程中，最简单、最正确的方法是利用“crontab -e”命令创建或编辑crontab文件。这个命令将会调用由EDITOR环境变量设定的文本编辑器。如果事先没有设置这个环境变量，crontab命令将会提示用户选用一个编辑器，如vim、ed或nano。

通常，无需具有超级用户的访问权限，任何用户都可以利用“crontab -e”命令，创建或编辑自己的crontab文件。如果crontab文件存在，则调用编辑器，打开现有的文件。如果crontab文件不存在，在写入并退出nano编辑器之后，创建一个新的crontab文件。创建的文件自动存储在/var/spool/cron/crontabs目录中，并以用户的注册名命名。为了使用vim编辑器，可以事先定义EDITOR变量。示例如下：

```
$ EDITOR=vi; export EDITOR
$ crontab -e
```

如果是超级用户，可以使用下列命令形式，为自己或其他用户创建或编辑一个crontab文件。注意，只有具有超级用户的特权，才能创建或编辑其他用户的crontab文件。如果未指定用户名，则创建或编辑自己的crontab文件，并以root命名。

```
# crontab -e [-u username]
```

输入上述命令之后，crontab将会调用事先设定或中间选定的编辑器。然后可以利用编辑命令，按照crontab文件的语法格式，把新增的指令加到crontab文件中。

下面以运行MySQL数据库备份的mysqldump命令为例，说明怎样在/var/spool/cron/crontabs目录中创建一个以gqxing用户身份运行的crontab文件。首先输入下列crontab命令：

```
$ crontab -e
```

然后利用编辑命令，把下列两行内容（其中第一行为注释）加到打开的临时文件中，使系统能够在每天的午夜23:55，利用mysqldump命令自动备份数据库、数据库表及其业务数据：

```
# Backup MySQL database at 23:55 everyday.
55 23 * * * gqxing mysqldump --add-drop-table -u gqxing -p1b2c3 books > /tmp/
books.`date '+%m%d'` >> /home/gqxing/backup/backup.log 2>&1
```

保存文件，退出编辑器。最终将会在/var/spool/cron/crontabs目录中创建一个名为gqxing的文件。

mysqldump命令后面的“>>/home/gqxing/backup/backup.log 2>&1”意味着把其标准输出及标准错误输出重定向到“/home/gqxing/backup/backup.log”日志文件中，从而把mysqldump命令的输出信息都记录下来，以备在上班时间随时查阅。

16.2.3 显示crontab文件

通常，普通用户不能直接访问/var/spool/cron/crontabs目录。为了检验系统中是否存在crontab文件，超级用户可以使用“ls -l”命令访问/var/spool/cron/crontabs目录，列出其中的文件。例如，下面的输出信息表明存在一个属于用户gqxing的crontab文件：

```
$ sudo ls -l /var/spool/cron/crontabs
总计 4
-rw----- 1 gqxing crontab 180 2008-11-14 11:19 gqxing
$
```

但是，任何用户均可使用“crontab -l”命令显示或检查属于自己的crontab文件。如同cat等命令能够显示任何文本文件的内容一样，“crontab -l”命令能够专门用于显示crontab文件的内容。且在输入这个命令时，无需指定crontab文件的路径名。

下面的例子说明了怎样使用“crontab -l”命令显示当前用户gqxing自己的crontab文件：

```
$ crontab -l
# Backup MySQL database at 23:55 everyday.
55 23 * * * mysqldump --add-drop-table -u gqxing -p1b2c3 books > /tmp/
books.`date '+%m%d'` >> /home/gqxing/backup/backup.log 2>&1
$
```

如果是超级用户，还可以利用下列命令形式，显示自己或其他用户的crontab文件。注意，只有具有超级用户的特权，才能查询其他用户的crontab文件。如果未指定用户名，则显示自己的crontab文件内容：

```
# crontab -l [-u username]
```

下面的例子说明了超级用户怎样显示其他用户的crontab文件:

```
$ sudo crontab -l -u cathy
# Backup MySQL dadabase at 23:55 everyday.
59 23 * * * mysqldump --add-drop-table -u cathy -pitsme books > /tmp/books.`date
'+%m%d'` >> /home/cathy/backup/backup.log 2>&1
$
```

16.2.4 删除crontab文件

通常, /var/spool/cron/crontabs目录的访问权限不允许普通用户直接删除crontab文件。为了删除自己的crontab文件, 可以采用下列命令:

```
$ crontab -r
```

上述命令只能删除属于自己的crontab文件, 且无需指定crontab文件的路径名。如果是超级用户, 可以利用下列命令, 删除自己或其他用户的crontab文件。注意, 只有具有超级用户的特权, 才能删除其他用户的crontab文件。如果未指定用户名, 则删除自己的crontab文件:

```
# crontab -r [-u username]
```

下面的例子说明了怎样使用“crontab -r”命令删除gqxing用户自己的crontab文件, 然后使用“crontab -l”命令, 验证crontab文件已经删除

```
$ crontab -r
$ crontab -l
no crontab for gqxing
$
```

16.2.5 crontab命令的访问控制

Ubuntu Linux系统通过选用的/etc/cron.deny文件, 或/etc/cron.allow文件限制用户使用crontab命令。只有这两个文件认可的用户才能够使用crontab命令创建、编辑、显示和删除自己的crontab文件。如果需要, 超级用户可以设置cron.deny或cron.allow文件, 限制或允许指定的用户使用crontab命令。注意, 只有超级用户才能够创建和访问cron.deny以及cron.allow文件。

cron.deny和cron.allow文件可以包含一系列用户名, 每个用户名占用一行。两个访问控制文件共同作用的结果如下:

- 如果cron.allow文件存在, 则只有这个文件中列出的用户才能够创建、编辑、显示和删除crontab文件。
- 如果cron.allow文件不存在, 则除了cron.deny文件中列举的用户之外, 其他用户均可创建、编辑、显示和删除crontab文件。
- 如果cron.allow或cron.deny文件均不存在, 任何用户都能够运行crontab命令。

初始安装Ubuntu Linux系统之后, 系统并不提供/etc/cron.allow和/etc/cron.deny文件。按照上述说明, 这意味着任何用户均可使用crontab命令。如果需要, 超级用户可以创建一个/etc/cron.deny文件, 增加用户名单, 禁止其使用crontab命令, 因而也就禁止其提交后台作业。

之后, 除了cron.deny文件中列出的用户之外, 其他任何用户均可运行crontab命令。如果创建了cron.allow文件, 则只有其中指定的用户才能够运行crontab命令。

因此, 为了禁止某些用户运行crontab命令, 可以直接编辑cron.deny文件, 增加禁止使用crontab命令的用户名单(每个用户名占一行)。例如, 下面给出的cron.deny文件表示不允许guest和visitor两个用户使用crontab命令:

```
$ cat /etc/cron.deny
guest
visitor
$
```

为了使指定的用户能够提交crontab作业, 也可以创建一个cron.allow文件, 首先把用户名root加到cron.allow文件中(如果不把root用户加到cron.allow文件中, 超级用户访问crontab命令时也会遭到拒绝); 然后再增加其他用户名, 每行一个, 加入的用户即可使用crontab命令。

为了验证普通用户是否能够访问crontab命令, 可以注册到相应的用户, 然后执行下列crontab命令:

```
$ crontab -l
```

如果用户能够访问crontab命令, 而且已经创建了crontab文件, 上述命令将会显示文件中的内容。否则, 如果相应的crontab文件不存在, 系统将会输出下列类似的错误信息:

```
$ crontab -l
no crontab for guest
$
```

上述信息说明, 当前的用户或者位于cron.allow, 或者不在cron.deny文件列举的名单中。

如果用户无权运行crontab命令, 不管相应的crontab文件是否存在, 将会显示下列出错信息, 意味着用户或者不在cron.allow文件列举的名单中, 或者位于cron.deny文件中:

```
$ crontab -l
You (guest) are not allowed to use this program (crontab)
See crontab(1) for more information
$
```

16.2.6 数据库定时备份实例

在MySQL数据库应用中, 为了能够在每天的指定时间定时备份数据库中的数据, 可以利用cron机制, 创建自己的crontab文件, 也可以在/etc/cron.d目录中创建一个新的crontab文件中, 增加一个数据库备份任务。

例如, 假定有一个MySQL数据库应用项目, 需要在每天晚上11点55分, 利用MySQL数据库的mysqldump命令备份一次数据。考虑到数据非常重要, 系统中至少应保留7天的备份数据。

为此, 可以利用crontab命令, 以用户gqxing的名义创建一个crontab文件, 在文件中增加一个定时执行MySQL数据库备份的Shell脚本, 以便在每天晚上11:55执行一次数据备份:

```
55 23 * * * /home/gqxing/script/sqldump > /dev/null 2>&1
```

其中, sqldump是一个Shell脚本, 其功能是以gqxing用户的身份, 执行mysqldump命令, 把gqxing用户的数据库、数据库表及其业务数据备份到books.mmd文件中。然后, 检查备份文件的数量, 如果超出7个文件, 则删除早期的备份文件。sqldump脚本的内容如下:


```
$ cat /home/gqxing/script/sqldump
#!/bin/bash
BAKDIR=/home/gqxing/backup
mysqldump --add-drop-table -u gqxing -p1b2c3 books > $BAKDIR/books.`date
'+%m%d'`
amount=`ls -l books* 2>/dev/null | wc -l`
if [ "${amount}" -gt 7 ]
then
    fname=`echo books*`
    set $fname
    mv -r $1
fi
exit 0
$
```

采用上述备份方法之后，将会在/home/gqxing/backup目录中保留7个数据库备份文件，例如：

```
$ ls /home/gqxing/backup
books.1215 books.1217 books.1219 books.1221
books.1216 books.1218 books.1220
$
```

16.3 调度一次性执行的作业

这一节开始介绍怎样使用at或batch命令执行下列任务：

- 提交at作业（命令或Shell脚本），使之在某个指定时间开始执行；
- 显示和删除已经提交的at作业；
- 控制用户是否能够使用at命令提交定时作业。

at命令的语法格式简写如下：

```
at [-mld] time [date]
```

其中，“-m”选项表示，一旦作业执行之后立即向用户发送电子邮件。time可以是1、2或4位数字，以时分形式（HHMM或HH:MM）指定作业开始运行的时间。如果指定的时间为整点时间（1或2位数字），分可以省略。如果按12小时制指定时间，时间后面应附加am或pm。其他可接受的关键字是midnight、noon和now。date是以月、日、年（MMDDYY、MM/DD/YY或MM.DD.YY）、“月名日（如June 1）”、星期几（如Monday）、关键字today或tomorrow等表示的日期。为了简化输入，也可以使用月、星期几或其他特殊关键字的前3个字符。

通常，任何用户均可以创建、显示和删除自己的at作业文件，但只有超级用户才有权访问其他用户的at作业文件。当利用at或batch命令提交一个at作业之后，系统将会赋予一个作业队列号码作为文件名的一部分，保留提交的at作业，以文件形式存储在/var/spool/cron/atjobs目录中，由atd守护进程负责处理以at或batch命令形式提交的作业。例如：

```
$ sudo ls -l /var/spool/cron/atjobs
总计 8
-rwx----- 1 gqxing daemon 3232 2009-11-15 12:26 a0000a0137f982
-rwx----- 1 gqxing daemon 3245 2009-11-15 12:49 b000100137f8e1
$
```


文件名的第一个字符表示作业的类型和状态, 其中“a”表示利用at命令提交的作业, “b”表示利用batch命令提交的作业。如果文件名的第一个字符变为等号“=”, 说明作业已经执行完毕, 暂时尚未清理。文件名的第2~6个字符以十六进制的数值形式表示作业的队列号。

例如, a0000a0137f982就是一个利用at命令提交的作业, 作业的队列号为10(0000a)。b000100137f8e1是利用batch命令提交的作业, 作业的队列号为16(00010)。

16.3.1 提交at作业

提交at作业包括三个要素: 输入at命令; 按照at命令的语法要求指定作业执行的时间; 输入准备执行的命令或Shell脚本。

为了提交一个at作业, 可在系统命令提示符下输入at命令, 同时指定作业的执行时间, 按下Enter键。在at命令提示符“at>”下, 输入准备执行的命令或Shell脚本。在输入命令或Shell脚本之后, 按下Ctrl-D组合键, 即可提交at作业。

例如, 下列命令提交的at作业将会在当天晚上6:30删除当前用户gqxing主目录中的core文件:

```
$ at 1830
warning: commands will be executed using /bin/sh
at> rm -r /home/gqxing/core > /dev/null 2>&1
at> <EOT>                                     (即按下Ctrl-D组合键, 下同)
job 18 at Fri Dec 11 18:30:00 2009
$
```

如果希望同时输入多个命令或Shell脚本, 每个命令或Shell脚本应占用一行, 并以Enter键结束。在输入所有命令或Shell脚本之后, 按下Ctrl-D组合键, 结束at命令的输入, 从而完成at作业的提交。

例如, 下列命令提交的at作业将会在11月30日午夜进入/home/gqxing/backup目录, 压缩其中的所有数据文件:

```
$ at 11:59 pm dec 25
warning: commands will be executed using /bin/sh
at> cd /home/gqxing/backup
at> bzip2 data*
at> <EOT>
job 19 at Fri Dec 25 23:59:00 2009
$
```

当需要提交一个作业, 但不关心究竟何时执行时, 可以使用batch命令。下面仍以上述的数据文件压缩为例, 说明怎样利用batch命令提交一个at作业, 使之压缩/home/gqxing/backup目录中的数据文件, 同时把开始执行时间和压缩完成时间等信息保存到一个日志文件bzip2.log中:

```
$ batch
warning: commands will be executed using /bin/sh
at> cd /home/gqxing/backup
at> date >> bzip2.log
at> bzip2 data*
at> date >> bzip2.log
at> <EOT>
job 20 at Fri Dec 11 16:52:00 2009
$
```

此外，还可以利用Here文档（参见第8章“Shell高级编程”），提供at或batch命令需要的输入数据。示例如下：

```
$ batch <<!  
> cd /home/gqxing/backup  
> date >> bzip2.log  
> bzip2 data*  
> date >> bzip2.log  
> !  
warning: commands will be executed using /bin/sh  
job 21 at Fri Dec 11 16:54:00 2009  
$
```



注意 如果执行命令或Shell脚本的输出信息很重要，可以使用I/O重定向的方式把输出信息写到一个文件中，以便将来能够查阅。

16.3.2 显示at作业及作业队列

同样，普通用户也不能直接访问/var/spool/cron/atjobs目录。为了查询已经创建的，且当前仍然位于at队列中的作业，可以使用atq或“at -l”命令。

在下面的例子中，atq命令列出了已经提交到at队列中的所有作业，以及at作业的执行时间信息：

```
$ atq  
19      Fri Dec 25 23:59:00 2009 a gqxing  
18      Fri Dec 11 18:30:00 2009 a gqxing  
$
```

上述命令也会显示已提交的at作业的状态信息。其中的a表示利用at命令提交的作业，b表示利用batch命令提交的作业，且两者尚未执行。

16.3.3 删除at作业

在at作业尚未执行之前，无需具有超级用户的特权，任何用户均可使用下列命令，从队列中删除自己的at作业。其中，“job-id”表示at作业的编号。

```
$ atrm job-id
```

或

```
$ at -d job-id
```

如果是超级用户，可以利用同样的命令，删除自己或其他用户的at作业。注意，只有具有超级用户的特权，才能删除其他用户的at作业。如果未指定用户名，则删除自己的at作业。

在下面的例子中，假定准备删除预定在11月30日午夜时分执行的at作业。首先，可以使用atq命令显示at作业队列，找出相应的作业号。接着，使用atrm命令从at作业队列中删除指定的作业。最后，验证指定的at作业是否已经删除。

```
$ atrm 19  
$ atq
```

```
18      Fri Dec 11 18:30:00 2009 a gqxing
$
```

删除指定的at作业之后，可以使用atq命令，验证指定的at作业是否已经删除（此时，删除的at作业不应出现）。

16.3.4 at命令的访问控制

Ubuntu Linux系统通过/etc/at.deny文件或选用的/etc/at.allow文件限制用户使用at系列命令。只有这两个文件认可的用户才能够使用at系列命令提交自己的at作业。如果需要，超级用户可以设置at.deny或at.allow文件，限制或允许指定的用户使用at系列命令。注意，只有超级用户才能够访问at.deny和at.allow文件。

at.deny和at.allow文件之于at系列命令，犹如crontab.deny和crontab.allow文件之于crontab命令，其组合作用的结果可以限定用户是否能够提交at作业等。

安装Ubuntu Linux操作系统之后，系统提供的at.deny文件包含一系列系统管理用户。这意味着超级用户与任何普通用户均可使用at系列命令，创建、删除或显示自己的at作业及其队列信息。必要时超级用户可以编辑这个文件，增加禁止使用at系列命令的用户名单，限制其使用at系列命令，禁止其提交at作业。

Linux系统通常不提供at.allow文件。因此，在安装Linux系统之后，除了at.deny文件中列出的用户之外，其他任何用户均可以提交at作业。如果创建了at.allow文件，则只有其中指定的用户才能够提交at作业。

/etc/at.deny文件由一系列用户名组成，每个用户占用一行。at.deny文件中列举的用户不能使用at系列命令提交at作业。下面是经过修改之后的at.deny文件，其中新增的guest用户不能再使用at系列命令，因而也就无法再提交定时执行的作业：

```
$ sudo cat /etc/at.deny
.....
guest
$
```

为了验证新加入/etc/at.deny文件中的用户是否能够使用at系列命令，可以使用su命令改变用户身份，然后运行atq命令，系统将会输出下列信息：

```
$ su - guest
Password:
$ atq
You do not have permission to use atq.
$
```

同样，如果用户试图提交一个at作业，其结果如下，说明guest属于限制用户：

```
$ at 10:30 am Saturday
You do not have permission to use at.
$
```

16.3.5 系统定时关机实例

为了在指定的时间关机，以应对用电高峰期间的停电或紧急系统维护，可以使用wall命令

提前向注册的用户发出停机通知，例如：

```
$ sudo wall <<!
> THE SYSTEM iscas WILL BE SHUT DOWN AT 16:30 ! ! !
> Please log off ASAP or risk your files being damaged.
> !

Broadcast Message from root@iscas
(/dev/pts/1) at 16:10 ...

THE SYSTEM iscas WILL BE SHUT DOWN AT 16:30 ! ! !
Please log off ASAP or risk your files being damaged.
$
```

然后利用at命令，按照预定的时间关闭系统，例如：

```
$ sudo at 1630
warning: commands will be executed using /bin/sh
at> poweroff
at> <EOT>
job 22 at Fri Dec 11 16:30:00 2009
$
```

输入poweroff命令，接着按下Enter键和Ctrl-D组合键之后，将会看到/var/spool/cron/atjobs目录中增加了一个at作业文件：

```
$ sudo ls -l /var/spool/cron/atjobs
总计 4
-rwx----- 1 root daemon 1992 2009-12-11 16:10 a0001d0137fee0
$
```

查阅上述文件可以看到，这个at作业文件主要包括三部分内容。其中，atrun一行主要说明这是一个at作业，执行作业的用户身份是超级用户（uid=0/gid=0）；从umask开始直至cd命令之前，主要是环境变量设置；最后一行即为提交的poweroff命令，示例如下：

```
$ sudo cat /var/spool/cron/atjobs/a0001d0137fee0
#!/bin/sh
# atrun uid=0 gid=0
# mail gqxing 0
umask 22
LS_COLORS=rs=0:di=01\;34:ln=01\;36:hl=44\;37:pi=40\;33:; export LS_COLORS
MAIL=/var/mail/gqxing; export MAIL
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/X11R6/bin; export PATH
LANG=zh_CN.UTF-8; export LANG
HOME=/home/gqxing; export HOME
LANGUAGE=zh_CN:zh; export LANGUAGE
LOGNAME=root; export LOGNAME
USER=root; export USER
USERNAME=root; export USERNAME
SUDO_COMMAND=/usr/bin/at\ 1630; export SUDO_COMMAND
SUDO_USER=gqxing; export SUDO_USER
SUDO_UID=1000; export SUDO_UID
SUDO_GID=1000; export SUDO_GID
```

```
cd /home/gqxing || {
    echo 'Execution directory inaccessible' >&2
    exit 1
}
poweroff
$
```

16.4 系统日志

在Ubuntu Linux系统的启动及运行过程中，系统发生的任何重要事件，都会记入各种日志文件中。对系统或应用程序进行诊断或排错时，这些信息具有非常重要的参考价值。如果了解每个日志文件的作用，很快就能够从中找出自己需要的信息。

利用各种日志文件，Ubuntu Linux系统提供了大量的系统事件、操作过程、说明信息以及错误信息。这些日志文件大多位于/var/log目录中，且是ASCII文本文件，采用标准的日志文件格式。此外，许多日志文件都是由系统日志守护进程rsyslogd根据系统和应用的配置要求生成的。而部分应用程序则生成并维护自己的日志文件，把需要记录的信息直接写入/var/log中某个子目录下的日志文件。

这一节简单介绍系统中的各种日志文件，以及其中包含何种日志信息。除了后续将要讨论的部分二进制日志文件，大多数日志文件均可利用文件显示命令，如less查阅其中的内容，也可以利用grep命令，按照关键字检索其中的特定信息。如果想要查询最早出现的信息，可以使用head命令；如果想要查询最新的日志信息，可以使用tail命令等。

如果想要实时监控某个日志文件，可以使用“-f”选项运行tail命令。例如，如果想要实时观测客户系统如何访问本地主机中的Apache服务器，可以使用下列命令（在客户系统的访问过程中，终端上将会断断续续地出现随时产生的访问信息）：

```
$ tail -f /var/log/apache2/access.log
```

16.4.1 系统日志文件

所谓的系统日志文件主要是指Ubuntu Linux的基本系统功能，如授权机制、守护进程以及系统消息等使用的日志文件，不包括附加的系统应用或用户应用提供的日志文件。

1. 授权日志

授权日志文件/var/log/auth.log用于跟踪记录各种授权机制（提示用户输入密码），如PAM插件认证技术、sudo命令、su命令以及OpenSSH远程注册等报告的授权信息。为了获取用户的注册过程、sudo和su命令等的执行情况，可以查阅这个日志文件。

2. 守护进程日志

守护进程日志位于/var/log/daemon.log文件，其中主要包含与系统守护进程以及应用守护进程（如GNOME显示管理器守护进程gdm、MySQL数据库守护进程mysqld）有关的信息。在获取各种守护进程的运行状况，对特定的守护进程排错方面，这个日志文件中的信息是非常有用的。

3. 调试信息日志

调试日志文件/var/log/debug能够提供详细的调试信息，其中包括来自Linux系统内核或应用程序输出的，经由rsyslogd记录的DEBUG级别的信息。在调试程序时，如调试硬件驱动程序或

应用程序时，其中的信息是非常有用的。

4. 系统内核日志

系统内核日志位于/var/log/kern.log文件，其中能够提供Linux系统内核报告的详细日志信息，包括系统配置信息，如CPU、内存、磁盘、I/O总线以及以太网卡等的配置信息。对于一个新系统，尤其是对用户定制系统的故障排除，这些信息是非常有用的。

5. 系统内核环形缓冲区

系统内核环形缓冲区并非实际存在的日志文件，而是位于系统内核的一个内存区，故无法直接查阅其中的信息。在系统的启动过程中，系统内核将会把引导过程中的软硬件模块初始化信息写到这个缓冲区中。若想观察其中存储的系统内核引导信息，可在注册后立即运行dmesg命令。如果引导过程中出现了问题，系统内核提供的引导信息有助于用户诊断系统问题，分析产生问题的原因。例如：

```
$ dmesg | less
```

dmesg命令主要用于考察和控制系统内核环形缓冲区，如显示其中存储的引导信息，清除过时的数据，以及设置缓冲区记录的信息级别等。实际上，/etc/init/dmesg.conf作业也是利用dmesg命令，把系统引导信息写入/var/log/dmesg日志文件中的。因此，也可以使用more或less等命令直接查询这个日志文件，获取系统引导过程中控制台上显示的信息。

6. 消息日志

消息日志文件/var/log/messages主要包含系统中各种实用程序及应用程序提供的一般性信息。为了考察实用程序及应用程序输出的，经由rsyslogd守护进程记录的各种INFO级别的信息，可以查阅这个日志文件。

7. 系统日志

顾名思义，系统日志文件/var/log/syslog是Linux系统中最大的信息集中存储位置，其中也许包含其他日志文件中没有保存的信息。如果从其他日志文件中找不到预期的信息，可以查询这个系统日志文件。

16.4.2 应用程序日志文件

在Ubuntu Linux系统中，除了上述的各种系统日志文件之外，还存在部分应用程序专用的日志文件。例如，/var/log/apache2目录中包含Apache服务器提供的日志文件，/var/log/samba目录中包含Samba服务器提供的日志文件。

1. Apache服务器日志

在Ubuntu Linux系统中安装Apache服务器时，通常会在/var/log目录中创建一个apache2子目录(/var/log/apache2)，其中包含两个具有不同用途的日志文件：

- /var/log/apache2/access.log——其中包含Apache服务器的客户系统访问记录。
- /var/log/apache2/error.log——其中包含Apache服务器报告的所有出错记录。

2. CUPS打印系统日志

通用UNIX打印系统(Common Unix Printing System, CUPS)采用/var/log/cups/error_log作为默认的日志文件，存储各种日志或错误信息。如果打印方面出现了问题，可以查阅或检索这个日志文件，从中找出端倪。

3. Samba服务器日志

Samba服务器是Linux与Windows系统之间常用的一种文件与打印资源共享技术。Samba在/var/log/samba目录中维护下列三种不同类型的日志文件

- log.nmbd——其中包含基于IP协议的NETBIOS网络通信方面的信息。
- log.smbd——其中包含Samba服务器启动, 以及SMB/CIFS文件与打印共享方面的信息。
- log.ip_address——用于记录来自特定客户系统的服务请求信息, 文件名中的ip_address是客户系统的IP地址, 如log.192.168.100.1。

4. X11服务器日志

在Linux系统中, 默认的X服务器是Xorg X11服务器, 如果计算机只配有一个显示器, 其日志文件为/var/log/Xorg.0.log。X11服务器的日志信息有助于诊断X11环境中出现的问题。

16.4.3 无法直接查阅的日志

/var/log目录中的部分日志文件是二进制数据文件, 不能直接查阅, 只能使用特定的程序读取其中的信息。

1. 注册失败日志

注册失败记录位于/var/log/faillog日志文件中, 这是一个典型的二进制数据文件, 不能直接读取, 需要时可利用faillog命令查阅。

2. 最近一次注册日志

最近一次(或者说最后一次)用户注册的时间记录位于/var/log/lastlog日志文件中。这也是一个典型的二进制数据文件, 需要使用lastlog命令查阅。

3. 用户注册日志

/var/run/utmp和/var/log/wtmp日志文件包含系统中每个用户的注册记录。在调度各种进程时, 系统将会自动记录每个进程的起止运行时间。对于当前活动的所有进程, 其相关信息均记录在/var/run/utmp文件中。当收到一个信号, 告知其调度的进程已经终止运行时, 如果/var/log/wtmp文件存在, 系统将会把相应的进程信息转录到wtmp文件中, 说明进程是什么时间启动和终止的, 以及终止的原因等。也就是说, /var/log/wtmp文件将会记录整个进程调度的历史。

/var/run/utmp和/var/log/wtmp文件中记录的部分信息如下(详细内容可见/usr/include/bits/utmp.h和/usr/include/bits/utmpx.h文件的定义或utmp/wtmp手册页):

- 进程或记录类型(其中, 1为运行级改变时间, 2为系统引导时间, 5为init调度的进程, 6为login进程, 7为用户进程, 8为终止的进程等);
- 进程ID;
- 缩写的终端设备名(如pts/2等);
- /etc/inittab文件的id字段(保留字段);
- 用户名;
- 远程系统名(远程注册时);
- 进程终止及出口状态;
- 时间记录(即进程或会话的起止运行时间);
- 远程系统的IP地址。

与/var/log/lastlog日志文件不同，/var/log/wtmp文件记录的是用户的注册与工作状态，如注册访问方式（终端设备名、远程系统名或IP地址等）、注册会话的起止运行时间等。利用who命令可以查询当前注册的用户信息，而利用last命令可以查询wtmp文件记录的用户注册历史等信息。如果再使用last命令的“-f”选项，指定/var/log/wtmp.N作为输入文件（其中N是一个从1开始编号的数字），还可以查询用户早期的注册历史等信息。例如：

```
$ last
gqxing pts/1      169.254.78.56    Sat Dec  5 11:40    still logged in
gqxing pts/1      169.254.78.56    Sat Dec  5 11:40 - 11:40 (00:00)
cathy  pts/0      :0.0             Sat Dec  5 10:55    still logged in
cathy  tty7      :0               Sat Dec  5 10:49    still logged in
reboot system boot  2.6.31-14-generi Sat Dec  5 10:47 - 11:40 (00:53)
.....
$
```

16.4.4 系统日志守护进程

系统日志守护进程rsyslogd是一个重要的守护进程，在系统的启动过程中自动启动，以后台方式运行。rsyslogd守护进程是一个总控开关，用于接收来自各种子系统或应用程序的信息，记录系统范围的日志信息，如果子系统或应用程序拥有自己专用的日志文件，则把信息送交子系统或应用程序特定的日志文件。考虑到系统的存储空间，每个日志文件均采用先进先出的方式，保存最新的日志信息。当日志文件超过其容量限制，过期的信息将会逐步移出日志文件。

rsyslogd记录的信息通常包含若干重要的通用字段，如记录时间、主机名以及信息的来源，如来自系统内核还是应用程序等。

1. 配置rsyslogd

rsyslogd采用模块化的设计，根据/etc/rsyslog.conf主配置文件及/etc/rsyslog.d目录中各个配置文件的定义，确定其支持的模块，以及如何记录系统日志。其中，/etc/rsyslog.conf主配置文件主要用于配置rsyslogd支持的模块，/etc/rsyslog.d目录中的配置文件主要用于分派每个日志文件应记录的数据内容及格式。如有必要，可以修改rsyslogd的任何配置文件，增加模块支持，定制日志文件记录的数据内容及其格式。/etc/syslog.d目录中的配置文件主要包含两个字段，其语法格式如下：

```
selector action
```

其中，selector是一个选择器，本身包含两部分数据：即写入日志文件的主体程序与优先级，中间以句点“.”作为分隔符。优先级也称日志信息级别，从低到高依次为debug（调试信息）、info（普通信息）、notice（提醒信息）、warning（警告信息）、err（错误信息）、crit（重要信息）、alert（警报信息）以及emerg（紧急信息）。设定的优先级越低，记录的日志信息越多。action动作字段用于定义日志文件，也即日志信息的目的存储位置，如标准的日志文件/var/log/syslog、专用的日志文件，以及把日志信息发送到指定主机名（如@somehost）的远程主机等。

rsyslogd配置文件的设置非常灵活，能够采用任何组合方式定义选择器与动作字段。下面是取自/etc/rsyslog.d/50-default.conf配置文件的部分内容：

```
$ cat /etc/rsyslog.d/50-default.conf
.....
```



```

auth,authpriv.*      /var/log/auth.log
*.*;auth,authpriv.none -/var/log/syslog
#cron.*              /var/log/cron.log
daemon.*             /var/log/daemon.log
kern.*               /var/log/kern.log
lpr.*                /var/log/lpr.log
mail.*               /var/log/mail.log
user.*               /var/log/user.log
.....
$

```

在上述配置中，优先权部分的星号“*”表示所有的信息级别，意味着应记录所有的日志信息；日志文件名前面的减号“-”表示禁止在每次写入日志文件之后立即执行内存与磁盘的数据同步。

2. 利用logger命令提交日志信息

Ubuntu Linux系统提供了一个日志工具logger，使用户能够把自己的信息随时加入系统日志文件/var/log/syslog。这是一个非常有用的工具，可用于系统维护脚本，如Perl或Shell脚本文件，提供标准的日志功能。logger命令的语法格式如下：

```
logger [-isd] [-f file] [-p pri] [-t tag] [-u socket] message
```

其中，“-s”选项表示既把指定的信息写到标准输出，也写入日志文件；“-t”选项表示在写入日志文件的每一行信息之前增加一个指定的标志；“-f”选项表示把信息写入指定的文件，而不是默认的日志文件/var/log/syslog；*message*表示写入日志文件中的信息。

例如，如果在命令行或脚本文件中使用下列命令，系统将会在运行期间把指定的信息写入/var/log/syslog日志文件：

```
$ logger This is a testing for user logging
$
```

使用下列命令，将会在/var/log/syslog日志文件中看到给定的信息及其运行时间：

```
$ grep testing /var/log/syslog
Dec  5 09:09:52 iscas gqxing: This is a testing for user logging
$
```

在Shell脚本中，更专业的方法是利用“-t”选项，在写入系统日志文件中的信息中增加一个标志字段（而非使用默认的用户名），标示信息的来源，以区别于其他日志信息。例如：

```
$ logger -t MyINFO This is a testing for user logging
$ grep MyINFO /var/log/syslog
Dec  5 09:11:57 iscas MyINFO: This is a testing for user logging
$
```

此外，还可以利用“-s”选项，在写入日志文件的同时，也在终端窗口显示给定的信息。例如，假定存在下列脚本：

```
$ cat ~/script/chkroot.sh
#!/bin/bash
# logger demo -- use logger utility in Shell script
#
```

```

logmsg="/usr/bin/logger -s -t MyScript "

# Print following message, write message to syslog in the same time
$logmsg "Root account status checking script."

# Test the status of root account on current machine
status=`grep root /etc/shadow | cut -d: -f2`
if [ "$status" = "*" ]; then
    $logmsg "The root account is locked."
else
    $logmsg "The root account has unlocked."
fi
$

```

取决于超级用户账号的封锁状态，执行上述脚本的结果示例如下（假定超级用户账号已经解锁）：

```

$ sudo ~/script/chkroot.sh
[sudo] password for gqxing:
MyScript: Root account status checking script.
MyScript: The root account is locked.
$ grep MyScript /var/log/syslog
Dec  5 09:13:58 iscas MyScript: Root account status checking script.
Dec  5 09:13:58 iscas MyScript: The root account is locked.
$

```

上述输出和日志文件的检索结果表示指定的信息既通过标准错误输出到终端窗口显示，也同时写入syslog日志文件中。

3. 日志文件的轮换

上述的任何日志文件filename.log，除了主日志文件之外，/var/log（及其子目录）中还包含filename.log.0、filename.log.1.gz、filename.log.2.gz以及filename.log.3.gz等形式的日志文件，这些文件是逐步轮换出的旧日志文件。也就是说，当超过预定的时间，系统将会自动删除最后一个文件，如filename.log.3.gz，然后依次重新命名日志文件，把filename.log.2.gz重命名为filename.log.3.gz，把filename.log.1.gz重命名为filename.log.2.gz，再利用gzip命令，把压缩后的filename.log.0重命名为filename.log.1.gz，把filename.log重命名为filename.log.0，最后重新创建新的主日志文件。日志文件的轮换与压缩，其主要目的是在节省磁盘空间的前提下，仍然保留一定的备份数据，以便用户能够查询其中的信息。

为了实现日志文件的轮换，Ubuntu Linux系统利用/etc/cron.daily/logrotate脚本（也可以利用/etc/cron.daily/syslogd脚本），由cron守护进程定时调用logrotate命令，logrotate命令再根据/etc/logrotate.conf配置文件或单独到/etc/logrotate.d目录中的其他配置文件（如apache2和mysql配置文件等）的定义，确定每个日志文件的轮换要求。

第17章 TCP/IP网络管理

网络与通信是操作系统的一个重要组成部分，Linux系统提供了丰富的网络通信功能。本章主要介绍IPv4（即传统的TCP/IP），以及网络接口、网关和路由等的设置，最后讨论TCP/IP网络的管理与维护。

17.1 网络接口设置

这一节将以以太网和ADSL/PPPoE网络为例，说明怎样设置TCP/IP。

17.1.1 以太网设置

在Linux系统中，网络接口的参数设置通常都是利用配置文件实现的。Ubuntu Linux系统采用/etc/network/interfaces配置文件定义每一个网络接口，因此所有的网络接口参数均位于interfaces配置文件中。系统启动脚本/etc/init.d/networking利用ifup和ifdown命令，依据配置文件中的定义，启动或关闭系统的网络功能。

为了设置或修改网络接口参数，可以使用编辑器，直接编辑网络接口的配置文件，也可以利用GNOME图形界面设置网络。

1. 网络接口配置文件interfaces

interfaces配置文件定义的网络接口参数可由若干“iface”、“mapping”、“auto”或“allow-”节组成。例如：

```
auto lo eth0
allow-hotplug eth1
iface lo inet loopback

mapping eth0
    script /usr/local/sbin/map-scheme
    map HOME eth0-home
    map WORK eth0-work

iface eth0-home inet static
    address 192.168.1.1
    netmask 255.255.255.0
    up flush-mail

iface eth0-work inet dhcp
iface eth1 inet dhcp
```

其中，网络接口前的关键字auto表示，当启动脚本/etc/init.d/networking采用“-a”选项运行ifup命令时，系统将会自动启用相应的网络接口。网络接口紧随auto关键字之后，位于同一行上。例如，“auto lo eth0”表示在启动系统时自动配置并启用环回接口以及第一个以太网接口。当配置文件包含多个auto节时，ifup命令将会按照列举的顺序，依次启用每一个网络接口。

关键字allow-auto和allow-hotplug等也表示自动启用相应的网络接口，但在执行诸如“ifup

—allow-hotplug eth0 eth1”形式的命令时，只能启用“allow-hotplug”后面列举的网络接口，如eth0或eth1。注意，“allow-auto”与“auto”的意义是相同的。

mapping用于定义网络的物理接口名与逻辑接口名的映射关系。mapping节的第一行由mapping和一个Shell模式组成，每个mapping节必须包含一个脚本定义，指定的脚本以物理网络接口名为参数。如果存在，随后的映射关系通过标准输入提交脚本。最终，脚本必须通过标准输出，输出一个表示网络逻辑接口名的字符串。

iface节用于定义一个逻辑网络接口名（如果mapping节不存在，逻辑接口名等同于物理接口名），其语法格式如下：

```
iface ifname inet-address-family method
```

其中，“ifname”表示逻辑网络接口名；“inet-address-family”是网络地址类型，表示支持的网络协议，如inet（IPv4）、inet6（IPv6）以及iipx等。“method”表示网络接口的配置方法，如loopback方法（用于定义IPv4环回接口）、static方法、dhcp方法、bootp方法（用于定义BOOTP服务器的IP地址），以及ppp方法等。

对于IPv4网络协议，如果需要，可在随后的附加行中定义IP地址、子网掩码以及广播地址等参数。此外还可以增加各种接口选项，表示启用还是禁用，以及如何启用或禁用相应的网络接口，例如，up command、pre-up command、post-up command、down command、pre-down command以及post-down command分别表示在网络接口启用时、启用前后、禁用时，以及禁用前后执行指定的命令。详见/usr/share/doc/ifupdown/examples/network-interfaces.gz文件给出的各种设置实例。

在IPv4网络协议中，可以采用各种方法配置网络接口。其中，static方法适用于静态分配网络接口的IP地址等参数，如IP地址、子网掩码、广播地址、网络地址、默认的网关、与网关的路由距离，以及MTU大小等。

• address address	# IP地址
• netmask netmask	# 子网掩码
• broadcast broadcast_address	# 广播地址
• network network_address	# 网络地址
• gateway address	# 网关地址
• metric metric	# 路由距离
• pointopoint address	# PPP协议IP地址
• mediatype	# 物理网络类型
• hw address class address	# 网络接口的MAC地址
• mtu size	# MTU的大小

例如，假定系统只配有一个以太网网络接口，为了采用static方法设置网路接口，interfaces文件可以配置如下：

```
$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
```

```

address 192.168.90.100
netmask 255.255.255.0
gateway 192.168.90.1
$

```

dhcp 方法适用于存在DHCP服务器的网络环境。系统可利用dhclient、pump、udhcp或dhcpcd等工具获取动态IP地址、主机名以及DNS服务器等。每个工具通常都有自己的配置文件。例如，dhclient使用的配置文件是/etc/dhcp2/dhclient.conf。

Ubuntu Linux系统主要采用dhclient实现IP地址的自动配置。动态主机配置协议（DHCP）的客户端软件dhclient能够利用DHCP或BOOTP协议，联系DHCP服务器，获取IP地址等网络接口参数，配置系统中的任何网络接口。当DHCP或BOOTP网络协议运行失败时，还能够静态地分配IP地址。

为了在办公室环境中采用dhcp方法设置网路接口，interfaces文件可以配置如下：

```

$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
$

```

ppp方法采用pon/poff命令配置PPP网络接口。在ppp方法中，唯一可用的选项为“provider name”。其中name是ISP的名字（取自/etc/ppp/peers目录中的文件），具体的配置步骤，可以参见17.1.2小节“ADSL网络连接”一节的讨论。

2. 使用GNOME图形界面设置网络接口

若想利用GNOME图形界面设置网络接口参数，可在GNOME桌面中选择“系统→首选项→网络连接”菜单，弹出如图17-1所示的“网络连接”对话框。

单击“添加”按钮增加网络设备，或用鼠标选中需要修改的网络设备名，然后单击“编辑”按钮。在弹出的“正在编辑Auto xxxx”对话框中，从“方法”下拉列表框中选择设置方法，如“自动（DHCP）”或“手动”。选择“手动”设置方法时需要自己输入IP地址、子网掩码和网关，以及设置DNS服务器地址等，如图17-2所示。



图17-1 “连接”对话框



图17-2 “正在编辑Auto xxxx”对话框

**注意**

系统默认的配置动作是采用DHCP协议配置网络接口，动态获取IP地址等网络参数。如果属于这种情况，无需设置网络接口。

完成上述配置过程之后，可以单击“应用”按钮，在“认证”对话框中输入sudo密码，最后单击“授权”按钮。保存网络参数设置，使系统在/etc/network目录中生成一个新的网络接口配置文件interfaces。

**注意**

不管是直接编辑配置文件，还是使用GNOME图形界面设置方式，若想让新的网络配置参数立即生效，都需要使用下列命令，重新启动/etc/init.d/networking脚本，或在系统启动过程中，利用/etc/rcS.d目录中的相应启动脚本（S40networking），按照interfaces配置文件的定义设置网络接口：

```
$ sudo /etc/init.d/networking restart
```

3. 网络接口配置命令ip

/etc/init.d/networking脚本利用ifup（或ifdown）命令和/etc/network/interfaces文件，配置每个网络接口。而ifup（或ifdown）命令底层采用的基本工具主要是ip、ifconfig和route等命令。

ip命令是Linux系统中的一个功能非常强大的工具软件，不仅可用于显示、设置网络接口设备，还可用于显示和设置路由等。其语法格式简写如下：

```
ip [ OPTIONS ] OBJECT { COMMAND | help }
    OPTIONS := { -s[tatistics] | -f[amily] { inet | inet6 ... } ... }
    OBJECT := { link | addr | route ... }
    COMMAND := { set | show | add | del ... }
```

为了便于理解，根据ip命令的功能，可以把ip命令分解为link（用于设置链路层的网络参数）、addr（用于设置网络层的参数，如IP地址等）和route（用于设置网络路由）三种常用类型的命令：

```
ip link show [ DEVICE ]
ip link set DEVICE { up | down | arp { on | off } |
    dynamic { on | off } | multicast { on | off } |
    address ADDR | broadcast ADDR | mtu MTU }

ip addr show [ dev DEVICE ]
ip addr { add | del } ADDR dev DEVICE [ broadcast ADDR ]

ip route show [ ROUTE ]
ip route { add | del | change | append | replace | monitor } ROUTE
```

其中，“-s”选项用于显示各种统计数据与时间信息。“-f”选项用于指定协议类型，如inet（IPv4）或inet6（IPv6）等。DEVICE表示网络接口的设备文件名。ADDR是网络接口的MAC地址（“ip link”命令）或分配给网络接口的IP地址（“ip addr”命令）。如果需要指定非标准的子网掩码，可在IP地址后附加一个斜杠字符“/”，然后给出子网的长度（二进制数值的位数）。ROUTE表示路由项。parameters是ip命令支持的各种参数。

例如，为了查询一个网络接口物理设备的工作状态，可以使用下列ip命令：

```
$ ip link show eth0
```

```

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 00:01:4a:03:c3:0c brd ff:ff:ff:ff:ff:ff
$

```

上述输出信息中的UP表示网络接口已经启用。如果尚未启用（其标志为DOWN），可以使用下列ip命令，启用指定的网络接口设备：

```

$ sudo ip link set eth0 up
$

```

为了设置并启用一个网络接口设备，可以使用下列ip命令：

```

$ sudo ip addr add 169.254.78.100 dev eth0
$

```

为了查询一个网络接口设备的工作状态，可以使用下列ip命令：

```

$ ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 00:01:4a:03:c3:0c brd ff:ff:ff:ff:ff:ff
    inet 169.254.78.100/24 brd 169.254.78.255 scope global eth0
    inet6 fe80::201:4aff:fe03:c30c/64 scope link
        valid_lft forever preferred_lft forever
$

```

为了查询网络的路由信息，可以使用下列ip命令：

```

$ ip route show
169.254.78.0/24 dev eth0 proto kernel scope link src 169.254.78.100
169.254.0.0/16 dev eth0 scope link metric 1000
default via 169.254.78.1 dev eth0 metric 100
$

```

4. 网络接口配置命令ifconfig

作为一个常规的网络管理工具，Linux系统仍然保留ifconfig命令。ifconfig命令主要用于配置网络接口的各种参数，包括IP地址、广播地址和网络掩码等。如果愿意，也可以在系统的生成过程中使用ifconfig命令配置系统中每个网络接口的IP地址。为了测试或维护网络接口，可以随时使用ifconfig命令，显示、修改或重新设置网络接口的IP地址等参数。注意，只有超级用户才能修改网络接口的配置参数。

ifconfig命令的一般语法格式如下：

```

ifconfig [-a] [interface]
ifconfig interface [addr_family] [address] [parameters]

```

其中，interface是网络接口的设备文件名。addr_family表示网络协议类型，如inet（IPv4）或inet6（IPv6）等。address是分配给网络接口的IP地址。parameters是ifconfig命令支持的各种参数，如表17-1所示。

在第一种语法格式中，“-a”选项用于显示系统中配置的所有网络接口。如果未提供任何选项，ifconfig命令只显示当前活动的网络接口的状态。如果指定了网络接口，ifconfig命令仅仅显示给定网络接口的状态。

表17-1 ifconfig命令支持的部分参数

参数	简单说明
up	启用网络接口，使得用户程序能够通过相应的网络接口访问TCP/IP应用和服务。在利用ifconfig命令为网络接口分配一个IP地址时，也借机启用相应的网络接口。当使用这个参数启用一个网络接口时，也会把相应的网络接口设置为UP和RUNNING状态。在使用down参数临时关闭一个网络接口之后，也可以使用这个参数重新启用网络接口
down	关闭网络接口，禁止任何程序通过相应的网络接口访问TCP/IP应用和服务，同时把网络接口标记为DOWN状态。注意，设置这个参数时也会自动地删除使用这个网络接口的所有路由表项
arp	默认情况下，启用地址解析协议（Address Resolution Protocol，ARP），实现网络地址与物理地址之间的映射，以便检测主机网络接口的物理地址。如果禁用ARP，ifconfig命令的输出信息中将会存在一个NCA RP标志
-arp	禁止使用ARP地址解析协议
netmask mask	指定网络接口使用的子网掩码。子网掩码是一个32位的二进制数值，其中1表示网络地址部分，0表示主机地址部分。其表示方法可以采用8个十六进制的数值，加上0x前缀；也可以采用4组十进制的数值，中间加句点“.”分隔符。注意，mask至少应当包括标准的网络地址部分
broadcast addr	指定网络的广播地址。默认的广播地址是网号不变，主机地址部分全为1的IP地址。如果设置了广播地址，同时也会设置网络接口的BROADCAST标志
pointopoint addr	启用点对点连接模式，这意味着把相应的网络接口直接连接到另一个主机的网络接口。pointopoint后面给出的是链路另一端的IP地址。在启用点对点连接模式的同时也设置网络接口的IFF_POINTOPOINT状态标志
mtu n	使用指定的数值，设置网络接口的最大传输单位（Maximum Transmission Unit，MTU），即一次数据传输最多能够处理的字符数量，这个数值也限制了分组数据的最大尺寸。不同类型的网络接口，MTU的上限值也不相同。例如，Ethernet默认的MTU的数值为1500
metric number	这个参数用于设置路由表项的度量值。路由信息协议（Routing Information Protocol，RIP）使用这个度量值建立路由表。ifconfig命令使用的默认度量值为0。如果没有启用RIP服务进程（如ripd），根本就不需要设置这个参数；即使启用了RIP服务进程，通常也不必关心这个度量值的设置
promisc	使用这个参数可以把相应的网络接口置于混杂模式。在一个广播模式的网络中，这种混杂方式使得网络接口能够收到所有的分组数据，而不管分组数据的最终目的地是否是网络接口所在的主机。这一技术使网络管理员能够利用分组数据过滤器或探测仪分析网络数据，从中找出网络问题。tcpdump等工具就是在此基础上实现的。这个参数也会同时设置网络接口的PROMISC状态标志
txqueuelen length	设置传输队列的长度

例如，为了手动设置主机系统中的以太网接口eth0，可以使用下列ifconfig命令：

```
$ sudo ifconfig eth0 192.168.90.100
$
```

为了检查刚才的设置是否已经生效，可以使用下列命令：

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:01:4a:03:c3:0c
```



```

inet addr:192.168.90.100 Bcast:192.168.90.255 Mask:255.255.255.0
inet6 addr: fe80::201:4aff:fe03:c30c/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1021 errors:0 dropped:0 overruns:0 frame:0
TX packets:558 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:105261 (105.2 KB) TX bytes:59007 (59.0 KB)

```

\$

17.1.2 ADSL网络连接

Ubuntu Linux系统也支持无线网络和ADSL拨号网络连接。为了配置基于PPPoE协议的拨号网络功能, 利用ADSL访问Internet, 可以采用pppoeconf命令, 在一系列对话框的引导下, 完成ADSL网络的设置。具体步骤如下:

(1) 运行pppoeconf命令, 当其找到以太网接口之后, 可以使用制表键选中“是”按钮, 按下Enter键。如图17-3所示, pppoeconf命令使用/etc/ppp/peers/dsl-provider文件存储ADSL网络配置信息。如果原有的数据需要保存, 可在此前进行备份, 否则选中“是”按钮继续, 如图17-4所示。



图17-3 以太网接口检测对话框

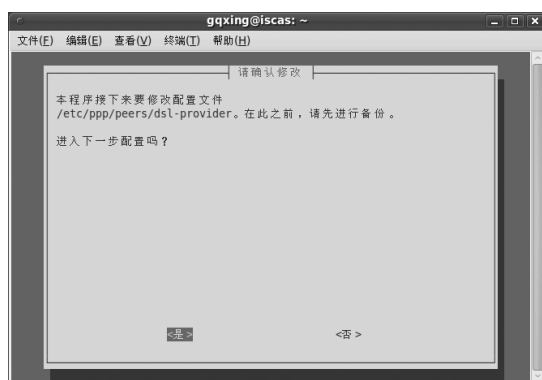


图17-4 备份提示对话框

(2) 在弹出的“经典模式”对话框中选择“是”按钮继续, 如图17-5所示。在图17-6所示的对话框中输入连接ADSL网络的注册用户名, 选中“确定”按钮继续。

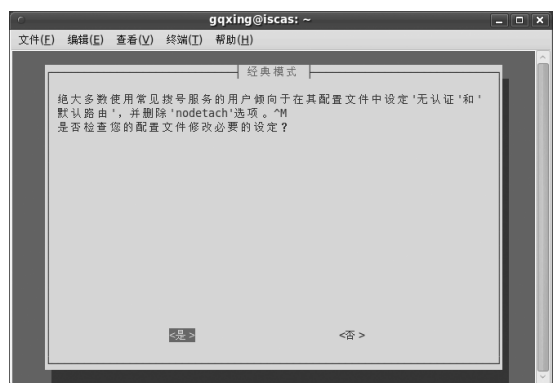


图17-5 “经典模式”对话框



图17-6 “输入用户名”对话框

(3) 在图17-7所示的对话框中输入连接ADSL网络的密码, 选中“确定”按钮继续。在图17-8所示的对话框中, 使用ISP提供的DNS, 选中“确定”按钮继续。



图17-7 “输入密码”对话框

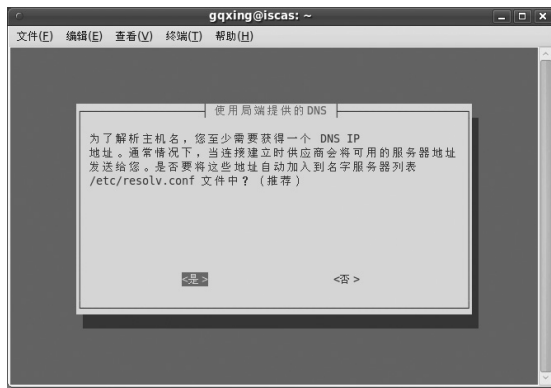


图17-8 DNS确认对话框

(4) 在图17-9所示的对话框中选择“是”按钮继续。在“完成”对话框中, 如果想要在每次启动系统时都要建立PPPoE连接, 利用提供的用户名和密码, 经过ADSL调制解调器, 连接Internet, 选择“是”按钮继续, 如图17-10所示。

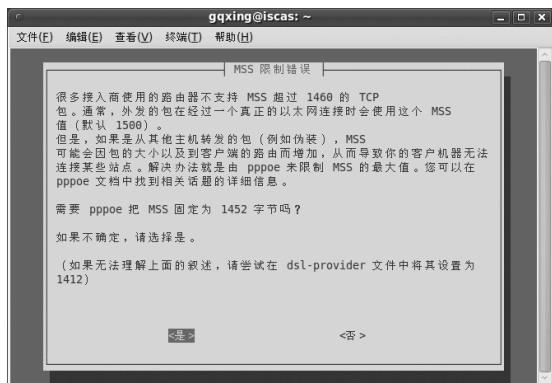


图17-9 “MSS限制错误”对话框



图17-10 设置完成对话框

(5) 此时, pppoeconf命令提示用户可以使用“pon dsl-provider”建立网络连接, 使用poff命令关闭网络连接。选择“是”按钮将会启动PPPoE网络, 如图17-11所示。启动PPPoE网络之后, 可以使用plog命令查询网络连接的状态, 或使用“ifconfig ppp0”命令查询网络接口的状态信息, 如图17-12所示。

(6) 在图17-13所示的界面中, plog和“ifconfig ppp0”命令的运行结果表示已经成功地建立了PPPoE网络连接, 此后即可使用浏览器访问Internet了。

在使用pppoeconf命令设置ADSL连接时, 系统将会在/etc/ppp/peers目录中自动创建一个dsl-provider配置文件, 其内容如下:

```
$ sudo cat /etc/ppp/peers/dsl-provider
# Minimalistic default options file for DSL/PPPoE connections

noipdefault
defaultroute
replacedefaultroute
```

```

hide-password
.....
noauth
persist
#mtu 1492
.....
plugin rp-pppoe.so eth0      # 使用以太网接口连接ADSL调制解调器
user "100001234567"         # ADSL网络注册的用户名。这里应为实际的注册用户名
usepeerdns
$

```



图17-11 建立连接对话框

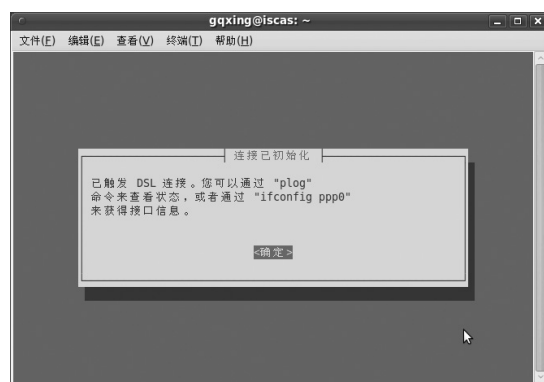


图17-12 连接完成对话框



图17-13 网络连接验证界面

而密码(包括注册用户名)等信息则存储在/etc/ppp/chap-secrets和/etc/ppp/pap-secrets文件中。其中, chap-secrets文件的内容如下:

```

$ sudo cat /etc/ppp/chap-secrets
# Secrets for authentication using CHAP
# client      server      secret          IP addresses
"100001234567" *          "password"
$

```

在作者配有一个以太网卡的系统中, 经过上述配置步骤之后, 其生成的interfaces文件(后4行)及resolv.conf文件内容如下:

```
$ cat /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet manual

auto dsl-provider
iface dsl-provider inet ppp
pre-up /sbin/ifconfig eth0 up # line maintained by pppoeconf
provider dsl-provider
$ cat /etc/ppp/resolv.conf
nameserver 202.106.195.68
nameserver 202.106.46.151
$
```

17.2 主机名字解析

在系统之间利用TCP/IP协议进行网络通信时，其中主要以IP地址标识数据的来源和目的。为了便于用户访问网络服务器，TCP/IP提供名字服务，以便实现从主机名到IP地址的转换。其中常见的有/etc/hosts文件、DNS和NIS等。

当存在多种主机名字的解析方式时，究竟采用哪一种方法解析主机名字呢？为此，Ubuntu Linux系统提供了一个/etc/hosts.conf文件，用于定义主机名字解析的顺序。文件中的每一行均以关键字开始，后面跟有适当的配置信息。其中一个常用的关键字为order，表示主机名字解析的顺序，后面跟有一个或多个名字解析方法，中间以逗号“,”作为分隔符。在Ubuntu Linux系统中，可以选用的解析方法是hosts和bind（即DNS）等，例如：

```
$ cat /etc/hosts.conf
# The "order" line is only used by old versions of the C library.
order hosts,bind
multi on
$
```

上述文件内容表示，在解析主机名字时，首先使用/etc/hosts文件。如果hosts文件不能满足要求，再使用DNS解析主机名字。

与DNS等名字解析方法相比，hosts文件相对比较简单。本章主要说明怎样使用hosts文件实现名字解析（有关DNS的介绍，详见《Ubuntu权威指南》一书）。

hosts文件的格式如下：

```
IP-address hostname [aliases...]
```

其中，IP-address为网络接口的IP地址，hostname是主机的名字，aliases是主机的别名。例如，下面是iscas系统中的一个hosts文件例子：

```
$ cat /etc/hosts
127.0.0.1          localhost
169.254.78.100     iscas
169.254.78.101     sinosoft
169.254.78.56      winxp          # ISP分配给Windows XP系统的IP地址
.....
$
```

17.3 网络路由设置

当网络中的所有主机均位于同一网段时，通过IP地址或主机名，系统之间可以随意进行访问。当准备访问的主机位于不同的网段时，则无法直达，必须通过网关或路由器才能访问。因此，为了访问外部网络，必须考虑路由设置。

按照网络的规模、网络拓扑结构的稳定性，以及主机在网络中扮演的角色——提供路由功能的主机或普通主机——可采用两种方式设置路由。

在一个大型的、拓扑结构经常发生变化的网络中，通常应当考虑使用动态路由。在一个小型或拓扑结构保持稳定不变的网络中，作为数据服务器或客户系统的普通主机，在完成本章前述的TCP/IP设置之后，必要时只需设置静态路由。考虑到路由的开销及其对计算机性能的影响，即使主机配有多个网络接口（如数据中心的数据库服务器），也以使用静态路由为最佳选择，动态路由可由专门的路由设备提供，或使用quagga等路由软件包实现。

在Ubuntu Linux系统中，可以采用route或ip等命令设置静态路由，其中最常用的基本工具仍然是route命令。这一节主要介绍如何利用route命令设置静态路由。

route命令的主要功能是维护网络路由表。利用route命令，可以增加、修改、删除、显示和监控路由表，而与增加或删除路由有关的route命令语法格式如下：

```
route [-v] [-A family] add | del [-net | -host | default] target
      [netmask Nm] [gw Gw] [[dev] if]
route [-v] [-A family] add default [gw Gw] [[dev] if]
```

其中，“-A family”表示地址类型，如net（IPv4）或net6（IPv6）等。“-host”表示把指定的目的地址强制解释为主机。“-net”表示把指定的目的地址强制解释为网络。target表示欲访问的目的主机或网络。Nm表示子网掩码。Gw表示通过哪一台主机（IP地址）或接口转发访问外部网络的数据。

假定本地系统的网络接口（其IP地址为192.168.90.101）连接到主机（其内部网络接口的IP地址为192.168.90.100），主机的另外一个网络接口（其IP地址为153.78.26.145）连接到外部网络。为了使本地系统能够访问153.78.26.0网络中的任何主机，可以使用下列命令，增加静态路由（参见图17-14）：

```
$ sudo route add -net 153.78.26.0 netmask 255.255.255.0 gw 192.168.90.100
$
```

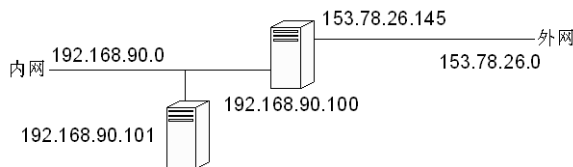


图17-14 增加静态路由示意图

为了利用路由器，访问路由器能够达到的任何网络和主机，最简单的方法是利用下列命令，增加默认的路由：

```
$ sudo route add default gw 192.168.90.100 dev eth0 metric 1000
$
```

上述设置方式只是临时性的，一旦重新启动系统，路由的设置也就不复存在了。在Ubuntu Linux系统中，为了永久性地设置静态路由，可以利用/etc/network/if-up.d/avahi-autoipd配置文件实现。

在系统的启动过程中，/etc/init.d/networking脚本将会运行avahi-autoipd脚本文件，根据其中的设置，利用route命令增加静态路由。因此，必要时可以把上述route命令加到avahi-autoipd文件中。



注意 在设置网络路由时，应尽量使用IP地址或网络接口的设备名，而不要使用主机名或网络名。

17.4 配置网络服务

当需要使用各种传统的网络服务时，需要用到inetd守护进程，设置其配置文件/etc/inetd.conf。inetd是一个超级的守护进程，负责集中管理许多标准的传统Internet服务，如telnet、ftp、finger和talk等，这些Internet服务的相应服务程序telnetd、ftpd、fingerd和talkd都是由inetd负责调度运行的。inetd同时支持UDP和TCP协议。

inetd是在系统启动过程中由/etc/init.d/openbsd-inetd脚本调度运行的。inetd守护进程用于监听Internet套接字的连接请求。当从套接字中发现一个连接请求时，inetd需要确定与套接字关联的相应网络服务，调用服务程序以响应连接请求。在服务程序结束运行之后，inetd将继续监听其负责管理的套接字。采用一个inetd守护进程调用多个服务程序的目的是为了减少系统的负载。

在开始运行之后，inetd首先会读取配置文件/etc/inetd.conf中的配置信息。当收到SIGHUP信号时，inetd守护进程将会重新读取inetd.conf配置文件。因此，在修改inetd.conf文件，增加、删除或修改服务程序之后，需要使用kill命令，向inetd守护进程发送SIGHUP信号，或重新启动inetd守护进程，才能使修改后的配置文件生效。为此，可利用pgrep、pidof或ps等命令找出inetd的进程ID号，再使用“kill -s SIGHUP”命令强制inetd重读其配置文件：

```
$ sudo kill -s SIGHUP `pidof inetd`
$
```

也可以利用下列命令，重新启动inetd守护进程：

```
$ sudo /etc/init.d/openbsd-inetd restart
* Restarting internet superserver inetd          [ OK ]
$
```

inetd.conf配置文件包含需由inetd守护进程调度运行的所有网络服务、服务程序的路径名及运行时使用的参数等，每个服务占用一行，其格式定义如下（字段之间以空格或制表符作为分隔符）：

```
srv_name socket_type protocol wait/nowait user srv_prog arguments
```

表17-2给出了inetd.conf配置文件每个字段的简单说明。

表17-2 inetd.conf文件的字段说明

字段名	简单说明
srv_name	/etc/inet/services文件中定义的有效服务名。当指定的服务是一个基于Sun-RPC的服务时，这个字段应是/etc/rpc文件中定义的一个有效服务名，同时附加一个斜杠“/”与RPC版本号后缀
socket_type	套接字类型可以是下列关键字之一： <ul style="list-style-type: none"> • stream：表示STREAMS套接字 • dgram：表示数据报套接字 • raw：表示原始套接字 • rdm：表示可靠的消息传递 • seqpacket：表示按序传输分组数据的套接字
protocol	/etc/inet/protocols文件中定义的有效协议，如tcp或udp等。除了协议，还可以采用“protocol [sndbuf=size][rcvbuf=size]”的形式指定发送和接收套接字缓冲区的大小。例如： <ul style="list-style-type: none"> • tcp.rcvbuf=16384 • tcp.sndbuf=64k • tcp.rcvbuf=64ksndbuf=1m 其中，k和m分别表示以1 KB和1 MB为单位
wait/nowait	这个字段的有效值是wait或nowait，说明当inetd调用相应的服务程序时是否需要等待服务进程运行终止。对于数据报类型的多线程服务进程，这个字段应为nowait。对于数据报类型的单线程服务进程，这个字段应为wait。采用TCP协议的服务程序通常应设为nowait。但是，如果单个服务进程需要处理多个连接请求，则应设为wait
user	指定一个用户名，表示以什么用户身份运行相应的服务程序。服务程序能够以普通用户的身份运行
srv_prog	指定服务程序的完整路径名，以便inetd能够正确地调用，使之执行请求的服务。如果请求的服务是由inetd本身提供的，这个字段应为internal
arguments	如果调用服务程序时需要提供命令行参数，必须在这个字段中指定整个命令行（包括命令名及其所有参数）。如果服务是由inetd内部提供的，这个字段应为internal

下面是一个inetd.conf配置文件的实例（注意，这是安装telnetd软件包时自动增加的内容，之前并不存在）：

```
$ cat /etc/inetd.conf
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
$
```

上述文件说明，telnet服务是基于TCP实现的，以telnetd的用户身份运行。in.telnetd服务程序的完整路径名为/usr/sbin/in.telnetd。一旦启用了in.telnetd服务程序，任何用户，只要有注册的账号，就可以利用telnet访问远程主机系统。

实际上，inetd.conf文件反映的是主机系统当前支持的各种Internet服务。如果想要封锁某个Internet服务，只需删除相应的服务项，或在服务项前增加一个注释符号“#”即可。

因此，为了确保数据库服务器或业务主机的系统安全，严禁任何用户使用telnet远程访问主机系统。可以利用编辑器，在inetd.conf文件的telnet定义前增加注释符号“#”，禁止inetd守护进程调用telnetd，封锁telnet服务，从而关闭主机系统的远程访问功能，防止telnetd响应用户的telnet访问请求。

17.5 网络管理与维护

对于新安装或首次接入网络的系统，如果根本就无法与网络中的其他任何主机进行通信，其主要问题也许是网络设置不当。如果主机一直工作正常，突然出现网络故障，那么网络接口卡故障可能是主要原因。如果主机能够与同一网络中的任何主机通信，但无法联系外部网络中的主机，问题肯定出在路由器上：可能主机的路由设置不当，或者路由器工作不正常。当然，也可能是网络的硬件连接存在问题。

如果网络通信存在问题，可以采用下列方法，利用ifconfig、netstat、ping等命令检测网络的硬件连接和软件配置，确定问题的原因。原因不明的问题轻则影响网络的性能，次则丢失数据，重则无法通信，无法传输任何数据。

- (1) 使用ping命令界定问题的性质，是丢失数据包，还是网络根本就不通；
- (2) 使用ifconfig命令，检查网络接口的状态信息，确定网络接口工作是否正常；
- (3) 使用netstat命令检查网络接口、路由表和协议统计数据等网络状态信息；
- (4) 检查hosts文件，确保文件中的内容是正确的；
- (5) 使用ps等命令，确定相关的服务进程（如sshd、vsftpd或telnetd等）是否已经启动。

17.5.1 使用ifconfig命令维护网络接口

利用ifconfig命令，可以手动地配置、修改或删除网络接口的IP地址，也可以使用ifconfig命令设置其他参数，获取网络接口的基本配置与状态信息等。一个简单的ifconfig查询（未指定任何选项和参数）能够给出如下信息：

- 系统配置的所有网络接口及其设备名；
- 网络接口的MAC地址以及链路协议封装类型；
- 赋予网络接口的IP地址、广播地址以及子网掩码等；
- 网络接口的初始化状态，如启用标志以及MTU设置等；
- 网络接口当前的运行状态，如分组与字节数据的收发统计等。

如果指定了网络接口的设备名，ifconfig命令将会给出特定网络接口的各种设置与状态信息。例如，下列ifconfig命令将会给出第一个以太网接口的相关信息：第一行是网络接口的链路协议封装类型（如Ethernet）与硬件MAC地址等；第二行是IP地址（如192.168.90.100）、广播地址（192.168.90.255）以及子网掩码（255.255.255.0）等；第三行是网络接口的状态标志，如启用标志（如UP等）、广播标志（如BROADCAST）以及MTU设置（如1500）等；从第四行开始是网络接口的各种统计数据，包括分组数据的收发统计及字节计数等。

```
$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:01:4a:03:c3:0c
          inet addr:192.168.90.100  Bcast:192.168.90.255  Mask:255.255.255.0
          inet6 addr: fe80::201:4aff:fe03:c30c/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1089 errors:0 dropped:0 overruns:0 frame:0
          TX packets:582 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:112506 (112.5 KB)  TX bytes:61594 (61.5 KB)
```

```
$
```


表17-3 给出了ifconfig命令输出字段的简单说明（有关RXpackets与TXpackets等分组数据统计信息字段的说明，详见表17-5）。

表17-3 ifconfig命令的部分输出字段及其说明

输出字段	屏幕显示	简单说明
	eth0	网络接口的设备名
Link encap	Bhernet	链路封装协议
HWaddr	00:17:31:C9:22:92	网络接口的MAC地址
inet addr	192.168.90.100	网络接口的IP地址
Bcast	192.168.90.255	广播地址
Mask	255.255.255.0	子网掩码
	UP	网络接口状态标志。表示网络接口当前是否已经启用，或是否已经初始化（UP或DOWN）
	BROADCAST	广播标志。表示网络接口是否支持广播
	RUNNING	传输标志。表示网络接口是否已经开始传输分组数据
	MULTICAST	多播标志。表示网络接口是否支持多播传输
MTU	1500	最大传输单位。表示网络接口的最大传输单位为1500个字节
Metric	1	度量值。这个数值主要供RIP建立网络路由表之用
RX bytes	23366 (22.8 KiB)	接收数据字节统计
TX bytes	24251 (23.6 KiB)	发送数据字节统计

当一个系统配置多个网络接口时，也可以使用“ifconfig -a”命令显示系统中配置的所有网络接口的状态信息，包括IP地址分配等：

```
$ ifconfig -a
eth0    Link encap:Ethernet  HWaddr 00:01:4a:03:c3:0c
        inet addr:192.168.90.100  Bcast:192.168.90.255  Mask:255.255.255.0
        inet6 addr: fe80::201:4aff:fe03:c30c/64 Scope:Link
        .....
eth1    Link encap:Ethernet  HWaddr 00:0e:35:cd:99:97
        inet6 addr: fe80::20e:35ff:fe0d:9997/64 Scope:Link
        .....
Lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        .....
$
```

17.5.2 使用netstat命令监控网络状态

netstat命令可以给出各种网络状态信息，包括网络接口设置、IP路由，以及各种网络协议的分类统计数据。根据选用的命令选项，netstat命令能够给出各种网络统计数据。netstat命令的基本语法格式简写如下：

```
netstat [-s] [-i [ifname]] [-r] [-n] [-atuwp] [-c] [delay]
```

其中，“-s”选项表示按照协议分类显示各种统计数据。“-i”选项用于显示网络接口的状态信息。ifname表示网络接口的设备名。“-r”选项用于显示核心路由表信息。“-n”选项表示只需显示IP地址，而不必把IP地址解析成相应的主机名或网络名。“-a”选项表示显示所有套接字的状态信息。“-t”选项表示仅显示TCP套接字的状态信息。“-u”选项表示仅显示UDP套接字的状态信息。“-w”选项表示显示原始套接字的状态信息。“-p”选项表示显示每个套接字所属程序的名字和进程ID。“-c”选项表示每秒一次，连续显示选定的信息。delay表示连续显示抽样统计数据的延迟时间或时间间隔（以秒为单位）。

1. 显示套接字的状态信息

默认情况下，netstat命令能够显示各种协议（如IPv4或UNIX等）已经打开的所有套接字的状态信息。例如：

```
$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 iscas.local:telnet      winxp:1034              ESTABLISHED
tcp        0      0 iscas.local:ftp         winxp:1132              ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags   Type       State      I-Node    Path
unix    20     [ ]     DGRAM          4654      /dev/log
unix     2     [ ]     DGRAM          2842      @/org/kernel/udev/udev
unix     2     [ ]     DGRAM          5039      @/org/freedesktop/hal/udev_event
.....
$
```

表17-4 给出了上述输出信息中部分字段的简单说明。

表17-4 netstat命令的部分输出字段及说明

输出字段	简单说明
Active Internet connections	
Proto	套接字连接采用的协议。如tcp、udp和raw
Recv-Q	本地主机接收队列中用户程序尚未读取的数据字节计数
Send-Q	本地主机发送队列中远程主机尚未确认是否已经读取的数据字节计数
Local Address	本地主机地址与端口号。除非使用了“-n”选项，主机地址通常为主机名或规范域名，端口号为相应的服务名。主机地址的表示形式为“主机:端口号”，或“网络:端口号”。其中，“主机”是源或目的主机的名字或IP地址（如/etc/hosts文件中定义的主机名或IP地址）。“网络”是源或目的网络的名字或网络地址（如/etc/networks文件中定义的网络名或网络地址）。“端口号”表示一个网络服务，既可以是/etc/services文件中定义的端口号，也可以是相应的服务名（如telnet）。在上述地址格式中，可以存在星号“*”通配符
Foreign Address	远程主机地址与端口号。表示方法同上
State	表示网络连接的状态。下面是可能出现的部分常见状态（注意，raw模式与UDP协议不提供网络连接的状态信息，故这一列通常为空）：

(续表)

输出字段	简单说明
CLOSED	网络连接已经关闭
CLOSE_WAIT	本地主机已经收到远程主机断开连接的请求, 且已经给予确认, 正在等待本地应用程序关闭连接
CLOSING	在经过LAST_ACK和TIME_WAIT两个状态之后, 本地与远程均进入CLOSING状态, 开始正式关闭网络连接
ESTABLISHED	网络连接已经建立, 可以双向传输分组数据
FIN_WAIT1	本地主机已向远程主机发送断开连接的请求, 正在等待远程主机的响应
FIN_WAIT2	本地主机在向远程主机发出断开连接的请求之后, 已经收到远程主机的确认
LAST_ACK	本地主机在收到应用程序关闭连接的请求之后, 再次向远程主机发出确认信息
LIS TEN	本地主机处于监听状态, 正在监听来自远程主机的连接请求
SYN_RECV	本地主机已经收到远程主机的连接请求, 且已经给予确认, 现正等待远程主机的最终确认
SYN_SENT	本地主机已经发出连接请求, 正在尝试建立套接字连接, 并等待远程主机的确认
TIME_WAIT	本地主机在第二次收到远程主机断开连接的确认之后, 接着向远程主机发出最终的确认, 然后开始着手关闭网络连接
Active UNIX domain Sockets	
Proto	套接字采用的协议(通常为unix)
RefCnt	引用计数, 表示加接到相应套接字的进程数量
Flags	标志字段。可能出现的标志包括SO_ACCEPTON (ACC)、SO_WAITDATA (W)和SO_NOSPACE (N)等
Type	Ubuntu Linux系统支持的套接字访问类型如下: SOCK_DGRAM 相应的套接字用于数据报(无连接模式) SOCK_STREAM 相应的套接字是一个Stream(连接模式)套接字 SOCK_RAW 相应的套接字用于原始套接字模式 SOCK_RDM 相应的套接字用于提供可靠的消息传递服务 SOCK_SEQPACKET 相应的套接字支持顺序分组数据传输 SOCK_PACKET 相应的套接字支持原始接口访问
State	状态字段通常包含下列关键字之一: FREE 相应的套接字尚未分配 LISTENING 相应的套接字正在监听一个连接请求 CONNECTING 相应的套接字连接正在建立 CONNECTED 相应的套接字连接已经建立 DISCONNECTING 相应的套接字正在断开连接 (空) 相应的套接字尚未完成连接
Path	套接字的路径名

如果想要查询与某个特定协议, 如TCP、UDP或原始套接字有关的状态或统计信息, 可以分别使用“-t”“-u”或“-w”选项。下面的例子说明了怎样使用“-t”选项专门显示与TCP协议有关的状态信息:

```
$ netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 iscas.local:telnet      winxp:1034              ESTABLISHED
tcp        0      0 iscas.local:ftp         winxp:1132              ESTABLISHED
$
```

2. 显示所有套接字的状态信息

使用netstat命令的“-a”选项，可以查询本地主机所有套接字（包括正在监听和尚未监听的套接字）的状态信息。下面是“netstat -a”命令的部分输出结果：

```
$ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:nfs                   *:.*                    LISTEN
tcp        0      0 *:58979                 *:.*                    LISTEN
tcp        0      0 *:swat                  *:.*                    LISTEN
tcp        0      0 localhost:mysql         *:.*                    LISTEN
tcp        0      0 *:43055                 *:.*                    LISTEN
tcp        0      0 *:sunrpc                *:.*                    LISTEN
tcp        0      0 *:42930                 *:.*                    LISTEN
tcp        0      0 *:ftp                   *:.*                    LISTEN
tcp        0      0 iscas.local:domain     *:.*                    LISTEN
tcp        0      0 localhost:domain       *:.*                    LISTEN
tcp        0      0 localhost:ipp           *:.*                    LISTEN
tcp        0      0 *:telnet                *:.*                    LISTEN
tcp        0      0 localhost:953          *:.*                    LISTEN
tcp        0      4 iscas.local:telnet     winxp:1034              ESTABLISHED
tcp        0      0 iscas.local:ftp        winxp:1132              ESTABLISHED
.....
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags   Type       State       I-Node   Path
unix    2      [ ACC ] STREAM LISTENING   10413    /tmp/orbit-cathy/linc-882-0-...
unix    2      [ ACC ] STREAM LISTENING   5634    /var/run/mysqld/mysqld.sock
unix    2      [ ACC ] STREAM LISTENING   8900    @/tmp/.ICE-unix/1988
.....
$
```

根据套接字的状态信息，可以确定某个网络服务进程是否已经启动。例如，为了查询是否已经启动了FTP服务器守护进程，可以使用下列命令：

```
$ netstat -a | grep ftp
tcp        0      0 *:ftp                   *:.*                    LISTEN
tcp        0      0 iscas.local:ftp        winxp:1132              ESTABLISHED
$
```

上述输出信息的状态字段LISTEN，说明FTP服务器守护进程已经启动，ESTABLISHED说明FTP服务器已经建立连接，TIME_WAIT说明FTP服务器正在等待用户输入命令。

3. 显示网络接口的状态信息，检测网络主机的可靠性

为了检测网络主机的可靠性和数据通信能力，可以利用netstat命令的“-i”选项，显示本地

系统网络接口的状态信息，借以确定系统发送和接收的分组数据数量。下面是“netstat -i”命令输出的一个例子，其中给出了网络接口的TCP/IP流量统计数据。

```
$ netstat -i
Kernel Interface table
Iface      MTU Met  RX-OK RX-ERR  RX-DRP RX-OVR    TX-OK  TX-ERR  TX-DRP TX-OVR  Flg
eth0       1500  0   1358     0      0      0     779     0      0      0  BMRU
eth1       1500  0     0      0      0      0      2     0      4      0  BMU
lo        16436  0    120     0      0      0     120     0      0      0  LRU
$
```

表17-5给出了“netstat -i”命令输出结果中部分字段的简单说明。

表17-5 “netstat -i”命令的部分输出字段及其说明

输出字段	简单说明
Iface	网络接口的名字
MTU	网络接口当前支持的最大传输单位。MTU是IP模块的设备驱动程序一次收发时能够处理的最大字节数量。对于以太网网络接口，MTU的默认值为1500。对于回环网络接口（loopback），其数值通常为8232。对于IEEE 802.3接口，其数值为1492
Met	网络接口当前支持的路由度量值
RX-OK	正确无误地接收了多少分组数据
RX-ERR	接收的分组数据本身有误的分组数据数量
RX-DRP	接收时由于各种原因而丢弃的分组数据数量
RX-OVR	由于接收错误和处理不及时等原因而遗失的分组数据数量
TX-OK	正确无误地发送了多少分组数据
TX-ERR	发送有误的分组数据数量
TX-DRP	发送时丢弃的分组数据数量
TX-OVR	由于发送错误而遗失的分组数据数量
Flg	网络接口设置的标志。可能出现的接口设置标志如下： <ul style="list-style-type: none"> • B: 广播地址设置标志 • L: 回环网络接口标志，表示相应的网络接口是回环接口 • M: 混杂接收模式标志，意味着网络接口能够接收所有的分组数据 • C: 网络接口禁用ARP标志 • P: 点到点链接标志，表示相应的网络接口采用的是点到点连接方式 • R: 网络接口已处于运行状态 • U: 网络接口已经启用

上述输出信息给出了当前主机中每个网络接口已经发送和接收的分组数据数量。利用发送和接收的分组数量，可以判定网络的运行是否正常。在一个正常运行的网络系统中，反映网络流量的RX-OK和TX-OK字段应当连续不断地增长。

例如，假定客户系统在尝试请求某个服务器引导自己的系统时，服务器接收的分组数据计数（RX-OK）不断增加，而输出的分组数据计数（TX-OK）保持不变。这一事实表明服务器正在接收和处理来自客户机含有引导请求的分组数据，但服务器不知道如何响应，故只有输入的请求，而服务器没有数据输出。这个结果说明网络的设置可能存在问题：可能是客户系统的配

置文件，如hosts文件中的服务器地址设置不正确，或者服务器的远程引导服务没有启动。

此外，如果RX-OK与TX-OK字段的计数一直保持稳定不变，说明系统根本没有接收和发送任何分组数据。这个结果表明：可能是网络连接存在问题，或者软件设置存在问题。总之，本地系统与其他系统之间没有任何数据通信。

4. 显示当前路由的状态

netstat命令的“-r”选项用于显示本地主机维护的路由信息，其中反映了本地主机已知的所有路由及其当前状态，示例如下：

```
$ netstat -r
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
169.254.78.0 * 255.255.255.0 U 0 0 0 eth0
link-local * 255.255.0.0 U 0 0 0 eth0
default 169.254.78.1 0.0.0.0 UG 0 0 0 eth0
$
```

表17-6 给出了“netstat -r”命令输出结果中部分字段的简单说明。

表17-6 “netstat -r”命令的部分输出字段及其说明

输出字段	简单说明
Destination	表示路由的目的主机或网络
Gateway	表示把分组数据转发到目的主机或网络需要用到的网关。如果这个字段为星号“*”，则意味着无需使用网关
Genmask	确定路由时使用的子网掩码。当给定一个IP地址，期望找到一个适当的路由时，系统内核将会利用这个子网掩码对IP地址进行逻辑与运算，然后依次检索每一个路由表项，从中找出匹配的路由
Flags	表示路由的当前状态。其中可能出现的状态标志如下： <ul style="list-style-type: none">· U: 表示相应的路由已经建立· G: 表示路由是利用网关实现的· H: 表示路由的目的地是一个主机，如回环网络接口表示的主机（127.0.0.1）· R: 表示路由是已恢复的动态路由· D: 表示路由是由路由守护进程或ICMP重定向消息动态建立的· M: 表示路由根据路由守护进程或ICMP重定向消息修改后而发生了变动· C: 表示缓存的路由
MSS	MSS（Maximum Segment Size）称做最大数据段尺寸，也是系统内核通过相应的路由能够构造和传输的最大数据报尺寸
Window	Window是本地系统能够接受远程主机一次传输的最大数据量
irtt	irtt是“initialround trip time”的缩写，意即初始的往返传输时间。TCP协议能够确保在通信的主机之间可靠地传输数据。如果数据传输有误，TCP将会重传丢失的数据报。TCP协议采用一个计数器，记录数据报传输远程主机花费的时间，以及收到一个确认需要多长时间，因此知道在重传数据报之前需要等待多长时间。这个时间称做数据报传输与确认的往返时间。在开始建立网络连接时，TCP协议将会采用一个初始的往返时间作为默认值。对于大多数网络而言，TCP协议采用的默认往返时间值是适当的，但对于一些速度较慢的网络，如果这个时间太短，将会引起不必要的数据报重传。必要的话，可以利用route命令设置irtt值。如果这个字段的值为0，则意味着采用默认值
Iface	表示路由经由的网络接口

5. 按协议显示统计信息

“netstat -s”命令用于显示不同协议（IP、UDP、TCP或ICMP等）的分类统计数据。利用这些统计数据，可以确定哪个网络协议存在问题。

```
$ netstat -s
Ip:
    1617 total packets received
    120 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    1497 incoming packets delivered
    961 requests sent out
Icmp:
    0 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
    0 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
Tcp:
    9 active connections openings
    2 passive connection openings
    8 failed connection attempts
    0 connection resets received
    2 connections established
    1101 segments received
    719 segments send out
    0 segments retransmited
    0 bad segments received.
    8 resets sent
Udp:
    246 packets received
    0 packets to unknown port received.
    0 packet receive errors
    246 packets sent
.....
$
```

利用“netstat -s”命令获取统计数据时，需要重点考察输入/输出错误、校验和错误、重传次数、分组数据丢失统计等字段。这些数据能够反映网络和主机的运行状态与性能，以及硬件连接和软件是否存在问题等。

17.5.3 使用ping命令测试远程主机的连通性

若想测试远程主机的连通性，最常见、最简单的方法是采用ping命令。运行时，ping命令将会利用ICMP协议向指定的远程主机发送ECHO_REQUEST请求信息，期望远程主机回以ECHO_REPLY响应信息。利用ping命令，可以检查与指定的远程主机是否已经建立了TCP/IP连接。ping命令的语法格式简写如下：

```
ping [-aAfn] [-c count] [-i interval] [-s size] [-w dead-line] dest-host
```

其中，“-a”选项表示每次响应时均给出声音警示。“-A”选项表示以实际的往返响应时间为间隔，连续发送数据报信息。“-f”选项表示连续不断地发送数据报信息，而不管是否收到响应信息。“-n”选项表示只需显示主机的IP地址，不必把IP地址解析成主机名。“-c”选项表示发送指定次数的数据报信息之后停止运行。“-i”选项表示发送每个数据报信息之间的时间间隔，默认值为1秒。“-s”选项表示分组数据的大小，默认值为56个字节（加上8个ICMP头字节，共64个字节）。“-w”选项表示以秒为单位的超时值。不管发送或接收了多少信息，一旦达到超时值，立即停止运行。dest-host是远程主机的名字。

1. 确定与远程主机的连通性

ping命令的主要用途是测试本地系统与远程主机之间的网络连通性，以及远程主机是否正在运行。如果指定的远程主机能够接收并响应ICMP请求，ping命令的运行结果如下：

```
$ ping iscas
PING iscas (192.168.90.100) 56(84) bytes of data.
64 bytes from iscas (192.168.90.100): icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from iscas (192.168.90.100): icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from iscas (192.168.90.100): icmp_seq=3 ttl=64 time=0.043 ms
Ctrl-C
--- iscas ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.039/0.045/0.054/0.008 ms
$
```

如果指定的远程主机关机，或收不到ICMP响应信息，ping命令的运行结果如下：

```
$ ping iscas
PING iscas (192.168.90.100) 56(84) bytes of data.
From iscas (192.168.90.100) icmp_seq=1 Destination Host Unreachable
From iscas (192.168.90.100) icmp_seq=2 Destination Host Unreachable
From iscas (192.168.90.100) icmp_seq=3 Destination Host Unreachable
Ctrl-C
--- iscas ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 3006ms,
pipe 3
$
```

2. 确定网络通信是否丢失分组数据

利用ping命令的“-A”选项，不仅可以确定本地系统与远程主机间的连通性，还可以检测两个主机之间的网络通信是否丢失分组数据。“-A”选项表示以实际的往返响应时间为间隔，向指定的远程主机连续发送分组数据，直至使用中断键终止命令的执行。对于普通用户而言，最小的往返响应时间间隔为200毫秒。示例如下：

```
$ ping -nA www.google.cn
PING cn.l.google.com (203.208.37.99) 56(84) bytes of data.
64 bytes from 203.208.37.99: icmp_seq=1 ttl=243 time=2.45 ms
64 bytes from 203.208.37.99: icmp_seq=2 ttl=243 time=3.38 ms
64 bytes from 203.208.37.99: icmp_seq=3 ttl=243 time=3.66 ms

--- cn.l.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 400ms
```



```
rtt min/avg/max/mdev = 2.458/3.167/3.660/0.516 ms, ipg/ewma 200.054/2.709 ms
$
```

“packet loss” 字段是分组数据丢失的统计数据。如果ping命令失败，或丢失的分组数据数量较大，需要利用ifconfig和netstat等命令进一步检查网络的工作状态。

17.5.4 使用ping命令检测网络主机的性能

ping命令不仅能够检测网络和主机的硬件连接，以及软件运行状态等连通性问题，还可用于检验网络和主机的响应能力与传输性能。

利用ping命令的“-f”选项，可以连续不断地向远程主机发送数据报信息，而且完全不必顾及是否收到响应信息。ping命令每发送一个ECHO_REQUEST信息，即在屏幕上输出一个句点“.”，本地系统每收到一个ECHO_REPLY响应信息，即在屏幕上输出一个退格符，擦除一个句点“.”。屏幕上输出的句点“.”多少可以大致说明远程主机的响应能力。按下Ctrl-C组合键后输出的“rtt min/avg/max/mdev”字段能够给出更准确的统计数据，说明往返传输时间的最小、平均以及最大时间值（毫秒），由此可以得到网络与主机的数据传输性能。示例如下：

```
$ sudo ping -f www.ubuntu.com
PING www.ubuntu.com (91.189.94.8) 56(84) bytes of data.
.. .....^C
--- www.ubuntu.com ping statistics ---
140 packets transmitted, 128 received, 8% packet loss, time 22659ms
rtt min/avg/max/mdev = 551.835/560.424/572.830/5.542 ms, pipe 47, ipg/ewma
163.019/555.472 ms
$ sudo ping -f 91.189.94.8
PING 91.189.94.8 (91.189.94.8) 56(84) bytes of data.
.....^C
--- 91.189.94.8 ping statistics ---
757 packets transmitted, 720 received, 4% packet loss, time 10436ms
rtt min/avg/max/mdev = 527.661/538.674/552.215/5.764 ms, pipe 45, ipg/ewma
13.804/543.616 ms
$
```

此外，如果使用“-s”选项指定较大尺寸的分组数据，还可以进一步检测网络和主机的响应能力与数据传输性能。

17.5.5 使用ftp命令检测网络主机的传输性能

利用ftp命令也可以检测网络主机的传输性能。进入ftp会话之后，可以使用下列ftp子命令，以/dev/zero作为输入，以/dev/null作为输出，传输一个较大的文件。避免使用硬盘（硬盘本身也有开销），也不必使用内存缓存整个文件，因而能够纯粹测试网络传输的性能。

```
put "|dd if=/dev/zero bs=32k count=10000" /dev/null
```

在下面的例子中，put命令最后一行的统计数据可以给出网络传输的速度。在执行实际的测试时，数据块的大小与传输的次数可由用户视具体情况而定：

```
$ ftp iscas
.....
ftp> put "|dd if=/dev/zero bs=32k count=10000" /dev/null
```

```
local: |dd if=/dev/zero bs=32k count=10000 remote: /dev/null
200 PORT command successful. Consider using PASV.
150 Ok to send data.
10000+0 records in
10000+0 records out
327680000 bytes (328 MB) copied, 4.84348 s, 67.7 MB/s
226 File receive OK.
327680000 bytes sent in 4.84 secs (66088.1 kB/s)
ftp> put "|dd if=/dev/zero bs=64k count=10000" /dev/null
local: |dd if=/dev/zero bs=64k count=10000 remote: /dev/null
200 PORT command successful. Consider using PASV.
150 Ok to send data.
10000+0 records in
10000+0 records out
655360000 bytes (655 MB) copied, 10.5424 s, 62.2 MB/s
226 File receive OK.
655360000 bytes sent in 10.54 secs (60708.1 kB/s)
ftp>
```

17.5.6 使用traceroute命令跟踪路由信息

在查询指定的主机时，traceroute命令利用IP协议的TTL（Time-to-Live）字段，要求途经的每个路由器或目的主机返回一个“ICMP TIME EXCEEDED”响应信息。在traceroute命令的跟踪过程中，TTL字段从默认值1开始，每经过一个中间路由器，TTL字段将会依次加1，直至收到一个“ICMP PORT UNREACHABLE”信息（表示已经到达指定的主机），或已达到TTL的最大值，跟踪过程立即结束。针对每一个TTL值，也即途经的每一个中间路由器，traceroute命令将会发送三个UDP数据报或“ICMP ECHO”查询，因而期望得到三个“ICMP TIME EXCEEDED”响应信息。

traceroute命令的主要功能是跟踪IP分组数据到达目的主机时经由的整个路径，用于显示相互通信的两个系统之间，IP分组数据从源主机到目的主机经过的所有中间路由。另外一个功能就是发现任何不合理的路由或路径不正确的路由。如果无法到达指定的主机，可以使用traceroute命令，检查IP分组通过什么路径联系远程主机，中间哪一个环节可能存在问题。

traceroute命令的语法格式如下：

```
traceroute [-dn] [-f first-ttl] [-w wait] [-m max-ttl] [-p port] [-q queries]
dest-host
```

其中，“-d”选项表示启用套接字级的调试功能。“-n”表示仅显示中间路由器或目的主机的IP地址，无需把IP地址解析成主机名。“-f”选项用于指定起始TTL值，默认值为1。“-w”选项表示等待中间路由器或目的主机返回ICMP响应信息的时间，默认值为5秒。“-m”选项用于指定TTL的最大值（也即最大的中转路由数量），默认值为30。“-p”选项表示traceroute命令使用的UDP端口号。“-q”选项表示针对每一个TTL，中间路由器或目的主机应返回几个ICMP响应信息，默认值为3。*dest-host*表示查询的主机。

traceroute命令将会显示到达目的主机期间途经的每一个路由器（包括目的主机）的IP地址，以及分组数据往返传输耗费的时间。这个时间信息对于分析两个主机之间任何环节的网络流量或通信能力是非常有用的。如果在发出UDP数据报或“ICMP ECHO”查询信息的5秒钟之内

一直没有接收到ICMP响应信息, traceroute命令将会在相应的中间路由器或目的主机的返回时间字段中输出一个星号“*”字符。

traceroute命令的每一行输出信息通常包含下列3个字段。

- TTL值(通常从1开始, 最大值为30);
- 中间路由器或目的主机的IP地址;
- 每一个ICMP响应信息的返回时间。

例如, traceroute命令的下列输出信息表明, 一个分组数据从本地主机到达远程主机www.google.cn, 中间跨越9个网络路径。这个输出信息同时也给出了分组数据途经每个网段时耗费的时间。

```
$ traceroute www.google.cn
traceroute to www.google.cn (203.208.37.160), 30 hops max, 60 byte packets
traceroute to www.google.cn (203.208.37.99), 30 hops max, 60 byte packets
 1  114.240.80.1 (114.240.80.1)  26.885 ms  26.979 ms  27.417 ms
 2  61.148.36.41 (61.148.36.41)  16.562 ms  18.372 ms  20.245 ms
 3  61.148.5.209 (61.148.5.209)  24.642 ms  25.762 ms  25.932 ms
 4  61.148.156.229 (61.148.156.229)  27.899 ms  29.739 ms  31.400 ms
 5  123.126.0.49 (123.126.0.49)  33.793 ms  35.906 ms  37.598 ms
 6  219.158.22.158 (219.158.22.158)  39.700 ms  25.469 ms  25.729 ms
 7  219.158.32.226 (219.158.32.226)  26.244 ms  23.750 ms  12.011 ms
 8  203.208.62.27 (203.208.62.27)  14.271 ms  15.922 ms  18.051 ms
 9  203.208.62.121 (203.208.62.121)  19.968 ms  21.872 ms  35.340 ms
10  bg-in-f99.1e100.net (203.208.37.99)  12.314 ms  14.379 ms  16.526 ms
```

第18章 TCP/IP网络应用

Linux系统提供的丰富网络功能有助于用户实现数据通信和文件传输。本章主要介绍TCP/IP网络应用，其中包括OpenSSH、Telnet和FTP等。

18.1 OpenSSH

在Linux系统中，OpenSSH是目前最流行的远程系统注册与文件传输应用，也是传统的Telnet、FTP与r系列网络应用的换代产品。其中，SSH（Secure Shell）可以替代Telnet、rlogin和rsh；SCP（Secure Copy）与SFTP（Secure FTP）能够替代FTP。

OpenSSH不仅适用于Linux系统，也可用于AIX、Solaris和HP-UX等UNIX系统，而且其SSH、SCP与SFTP等客户端软件也可用于Windows系统。如有必要，Windows用户可以下载Windows版的OpenSSH客户端软件，在Windows系统与Linux系统之间实现文件的复制。

OpenSSH采用密钥的方式对数据进行加密，确保数据传输的安全。在正式开始传输数据之前，双方首先要交换密钥。当收到对方的数据时，再利用密钥和相应的程序对数据进行解密。这种加密的数据传输有助于防止非法用户获取数据信息。

OpenSSH采用随机的方式生成公私密钥。密钥通常只需生成一次，必要时也可以重新制作。

当使用ssh命令注册到远程系统时，OpenSSH服务器的sshd守护进程将会发送一个公钥，OpenSSH客户端软件ssh将会提请用户确认是否接受发送的公钥。同时，OpenSSH客户机也会向服务器回送一个密钥，使OpenSSH连接双方的每个系统都拥有对方的密钥，因而能够解密对方经由加密链路发送的加密数据。

OpenSSH服务器的公钥与私钥均存储在/etc/ssh目录中。在OpenSSH客户端，用户收到的所有公钥，以及提供密钥的OpenSSH服务器的IP地址均存储在用户主目录下的~/.ssh/known_hosts文件中（注意，.ssh是一个隐藏的目录）。如果密钥与IP地址不再匹配，OpenSSH将会认为某个环节出了问题。例如，重新安装操作系统或升级OpenSSH时都会导致系统再次生成新的密钥。当然，恶意的网络攻击也会造成密钥的变动。因此，当密钥发生变化时，应当了解密钥发生变化的原因，以确保网络访问期间的数据安全。

18.1.1 安装OpenSSH服务器

在Ubuntu Linux系统的安装过程中，作为一个基本的系统软件，OpenSSH的客户端软件包将会随着Linux系统一起安装。但OpenSSH服务器需要单独安装。为了安装openssh-server服务器软件包，可以使用apt-get、aptitude或synaptic等软件工具，例如：

```
$ sudo aptitude install openssh-server
```

安装之后，为了验证OpenSSH服务器的sshd守护进程是否已开始运行，可以使用下列命令（参见第10章“进程管理”）：

```
$ pidof sshd
5568
$
```

18.1.2 sshd_config配置文件

/etc/ssh/sshd_config是OpenSSH服务器的默认配置文件，sshd守护进程根据其中的定义，规范其处理动作。如果需要，也可以在sshd的命令行中使用“-f”选项指定其他配置文件。

sshd_config配置文件的每一行或者是一个注释行（起始字符为注释符号“#”），或者是一个参数设置。sshd_config配置文件的语法格式如下：

```
parameter value
```

表18-1 给出了部分重要的配置参数及简单说明。

表18-1 sshd的部分配置参数及说明

配置参数	简单说明
AcceptEnv	指定客户端发送的哪些环境变量能够复制到当前会话的运行环境（客户端需要设置其配置文件ssh_config中的SendEnv参数）。注意，只有SSH协议版本2支持环境变量的传递。设置时只需指定变量的名字，其中可以包含通配符星号“*”与问号“?”。指定多个环境变量时，中间需加空白字符。此外，还可把环境变量分布到多个AcceptEnv配置参数中。需要注意的是，某些环境变量会对限制用户的运行环境带来负面影响，因此，应当仔细地使用该配置参数。这个配置参数的默认值是拒绝接受任何环境变量
AddressFamily	指定sshd守护进程支持的地址系列，也即网络协议，其有效值是any、inet（仅支持IPv4）或inet6（仅支持IPv6）。默认值是any，表示同时支持两种网络协议
AllowGroups	设定这个配置参数时，可以列举多个用户组的名字或模式，中间加空格分隔符。如果设定了这个配置参数，只允许其用户组（包括附加用户组）匹配指定用户名或模式的用户注册。注意，这里只能指定用户组名，不能使用用户组ID。通常，所有用户组的成员均允许注册。sshd将会按照Deny-Users、AllowUsers、Deny-Groups与AllowGroups的顺序依次处理用户注册的限定策略
AddressFamily	指定sshd使用哪一种网络协议类型，其有效值包括any、inet（仅支持IPv4）或inet6（仅支持IPv6），默认值为any
AllowTcpForwarding	指定是否启用TCP转发功能，其默认值为yes。注意，禁止TCP转发功能并不能改善系统的网络安全问题，除非也拒绝用户访问Shell，否则用户总是能够利用Shell安装自己的转发软件
AllowUsers	设定这个配置参数时，可以列举多个用户名或模式，中间加空格分隔符。如果设定了这个配置参数，只允许匹配指定用户名或模式的用户注册。同样，这里只能指定用户名，不能使用用户ID。通常，所有用户均可注册。如果用户名模式采用USER@HOST形式，需要单独检测USER与HOST，从而能够确保特定主机中的特定用户注册
AuthorizedKeysFile	指定包含用户认证公钥的文件。AuthorizedKeysFile配置参数的值可以包含%I形式的标记符，这些标记符能够在建立连接期间予以替换。其中，%I是单个百分号“%”，%h可以替换成认证用户的主目录，%u可以替换为注册用户的用户名。经过上述替换后，AuthorizedKeysFile指定的文件将会成为一个绝对路径名，或相对于用户主目录的相对路径名。默认值是ssh/authorized_keys

配置参数	简单说明
ChallengeResponseAuthentication	指定是否允许使用提示/应答式认证，默认值为no。sshd支持login.conf文件中定义的所有认证类型
DenyGroups	设定这个配置参数时，可以列举多个用户组的名字或模式，中间加空格分隔符。如果设定了这个配置参数，则禁止其用户组（包括附加用户组）匹配指定用户组名或模式的用户注册。同样，这里只能指定用户组名，不能使用用户组ID。通常，所有用户组的成员均允许注册
DenyUsers	设定这个配置参数时，可以列举多个用户名或模式，中间加空格分隔符。如果设定了这个配置参数，则禁止匹配指定用户名或模式的用户注册。同样，这里只能指定用户名，不能使用用户ID。通常，所有用户均可注册。如果用户名模式采用USER@HOST形式，需要单独检测USER与HOST，从而能够确保正确地限制特定主机中的特定用户注册
HostKey	指定SSH使用的包含主机私钥的文件，默认值是/etc/ssh/ssh_host_key（协议版本1），以及/etc/ssh/ssh_host_rsa_key与/etc/ssh/ssh_host_dsa_key（协议版本2）。注意，sshd将拒绝使用一个同组用户或其他任何用户均可访问的文件。注意，SSH允许采用多个主机私钥文件
ListenAddress	<p>指定sshd应当监听的本地IP地址。在指定监听地址时，可以采用下列形式：</p> <ul style="list-style-type: none"> • ListenAddress host IPv4_addr IPv6_addr • ListenAddress host IPv4_addr:port • ListenAddress [host IPv6_addr]:port <p>其中，host表示主机名，IPv4_addr表示IPv4地址，IPv6_addr表示IPv6地址，port表示端口号。如果ListenAddress配置参数中未指定端口（第一种定义形式），sshd将会监听指定的地址及其所有端口，除非Port配置参数之前另有限定。因此，任何Port配置参数（如果存在）必须位于未加端口限制的ListenAddress配置参数之前。在sshd_config配置文件中，允许同时指定多个ListenAddress配置参数。通常，sshd将会监听本地系统的所有IP地址</p>
LoginGraceTime	如果用户未能成功地注册，sshd将会在这个配置参数指定的时间过后断开连接。如果参数值为0，则表示没有时间限制，默认值为120秒
LogLevel	指定sshd的日志信息级别，可取的参数值是SILENT、QUIET、FATAL、ERROR、INFO、VERBOSE、DEBUG、DEBUG1、DEBUG2和DEBUG3，默认值是INFO。DEBUG与DEBUG1是等同的，DEBUG2与DEBUG3表示记录较高级别的调试信息。采用DEBUG级别有可能会侵犯用户的隐私，因而不建议使用
MaxAuthTries	指定每个连接请求容许的最大认证尝试次数。一旦注册失败的次数达到这个数值的一半，之后再出现的失败尝试将会记录在日志文件中。这个配置参数的默认值是6
PasswordAuthentication	指定是否允许使用密码认证，默认值是yes
PermitEmptyPasswords	当采用密码认证方式时，指定sshd是否允许使用空的密码注册，默认值是no
PermitRootLogin	指定超级用户root是否能够使用ssh注册。可选的参数值必须是yes、without-password、forced-commands-only或no之一，默认值是yes。如果把这个配置参数设置为without-password，意味着禁止root用户采用密码认证。如果设为forced-commands-only，则允许root用户使用公钥认证方式注册，但仅当在命令行的“-o”选项中指定了公钥认证（PubkeyAuthentication）。对于远程备份而言，这是非常有用的，即使在禁止root注册的情况下亦然。此时，禁止root用户使用其他认证方法。如果把这个配置参数设置为no，表示禁止root用户注册

(续表)

配置参数	简单说明
PidFile	指定包含sshd守护进程PID的文件，默认值是/var/run/sshd.pid
Port	指定sshd监听的端口号，默认值是22。配置文件允许同时指定多个Port配置参数，参见ListenAddress配置参数的说明
PrintLastLog	指定以交互方式注册时，sshd是否应当输出用户上一次注册的日期和时间，默认值是yes
PrintMotd	指定当用户以交互方式注册时，sshd是否应当显示/etc/motd文件中的内容，默认值为no。在有些Linux系统中，这一处理动作是由Shell或/etc/profile实现的
Protocol	指定sshd支持的OpenSSH协议版本，可取的值是1和2，默认值为2。如果同时指定多个版本号，之间需加逗号“,”分隔符
PubkeyAuthentication	指定是否允许采用公钥认证，默认值是yes。注意，这个配置参数仅适用于协议版本2
RSAAuthentication	指定是否允许使用纯RSA认证，默认值是yes。这个配置参数仅适用于SSH协议版本1
StrictModes	指定sshd是否应在接受用户注册之前检测文件的访问权限，以及用户的文件与主目录的属主等属性。默认值是yes
Subsystem	用于配置一个外部服务程序，如文件传输服务器sftp-server。配置参数的值应是一个系统名与命令（如果命令需要选项与参数，必须同时给出），能够基于客户系统的请求开始运行，其默认值为“sftp /usr/lib/openssh/sftp-server”。sftp-server命令实现了sftp文件传输子系统。注意，这个配置参数仅适用于OpenSSH协议版本2
TCPKeepAlive	指定sshd是否向客户系统发送TCP keepalive消息。如果发送此消息，服务器便能够及时了解客户系统的网络连接状态、关机或系统崩溃等情形。但这也意味着，如果网络路由临时失灵，导致网络连接暂时断开，有可能会引起不必要的麻烦，而这也并非设置此参数之本意。从另一方面讲，如果不发送TCP keepalives消息，sshd可能会无限期地维持当前的网络会话，空耗服务器的资源。这个配置参数的默认值是yes，即发送TCP keepalive消息，以便服务器能够及时了解网络连接是否已经断开，客户系统是否已经关机等，从而避免无限期地维持无效的网络会话。为了禁止发送TCP keepalive消息，可以把这个配置参数设为no
UseLogin	指定login命令是否可用于交互式注册会话，默认值是no。注意，login绝对不能用于远程命令执行
UsePAM	启用PAM插件式认证模块。如果把这个参数设置为yes，除了PAM账号与会话模块之外，PAM认证还能够使用ChallengeResponseAuthentication与PasswordAuthentication模块处理所有的认证类型。由于PAM提示/应答式认证基本上等同于密码认证，故应当禁用PasswordAuthentication或ChallengeResponseAuthentication配置参数。如果启用了UsePAM，则不能以普通用户的身份运行sshd。这个配置参数的默认值为no

下面是取自/etc/ssh/sshd_config配置文件的部分内容:

```
$ cat /etc/ssh/sshd_config
.....
```



```
# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
.....
# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile      %h/.ssh/authorized_keys
.....
Subsystem sftp /usr/lib/openssh/sftp-server
UsePAM yes
$
```

根据表18-1中的说明，以及Ubuntu Linux系统提供的/etc/ssh/sshd_config配置文件可知，sshd将会监听本地系统所有IP地址的22号TCP端口，支持OpenSSH协议版本2，允许超级用户root注册，采用PAM插件式认证模块，支持公钥认证与纯RSA认证，采用单独的子系统sftp，采用sftp-server作为服务器，实现文件的安全传输。

18.1.3 使用SSH注册到远程系统

在OpenSSH中，ssh是一个重要的客户端应用程序。利用ssh，可以其采用加密的通信方式，注册到远程系统。ssh命令的语法格式简写如下：

```
ssh [-l login_name] [-p port] [user@]hostname [command]
```

其中，“-l”选项用于指定用户名，表示以哪一个用户身份注册到远程系统。如果不提供用户名，则以当前用户的身份注册到远程系统。“-p”选项用于指定TCP端口。例如，下列命令形式表示仍以本地系统的gqxing用户身份，采用默认的端口22，注册到远程系统（注意，从“Linuxiscas……”一行开始，直至“http://help.ubuntu.com/”一行为止，这些内容出自/etc/motd文件，之后的例子中均省略）：

```
$ ssh iscas
gqxing@iscas's password:
Linux iscas 2.6.31-16-generic #52-Ubuntu SMP Thu Dec 3 22:00:22 UTC 2009 i686

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/

Last login: Fri Dec 11 20:22:48 2009 from sinosoft
$
```

利用“-l”选项，可以使用不同的用户身份注册到远程系统。例如，下列命令意味着本地系统的当前用户以cathy的用户身份注册到远程系统iscas：


```
$ ssh -l cathy iscas
cathy@iscas's password:
.....
Last login: Wed Dec 16 18:45:23 2009 from sinosoft
$
```

除了“-l”选项之外，还可以采用“user@hostname”形式，以指定用户的身份注册到远程系统：

```
$ ssh gqxing@iscas
gqxing@iscas's password:
.....
Last login: Wed Dec 16 15:45:11 2009 from sinosoft
$
```



第一次使用ssh注册到远程系统时，ssh会发出一个警告信息，提示用户确认连接的远程系统是否正确。如果用户回答yes，ssh将会在用户主目录的~/.ssh/known_hosts文件中存储远程系统的密钥，同时也会把客户端用户的密钥发送到远程系统：

```
$ ssh iscas
The authenticity of host 'iscas (169.254.78.100)' can't be established.
RSA key fingerprint is 53:e1:e2:3f:c4:af:99:74:78:05:7a:dc:c2:25:4f:4c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'iscas,169.254.78.100' (RSA) to the list of known hosts.
gqxing@iscas's password:
Linux iscas 2.6.31-16-generic #52-Ubuntu SMP Thu Dec 3 22:00:22 UTC 2009 i686
.....
Last login: Fri Dec 11 12:22:03 2009 from sinosoft
$
```

此后，当用户再次注册到同一远程系统时，就不会再出现上述提示信息了。

18.1.4 执行远程系统命令

除了类似于Telnet的远程终端仿真功能之外，ssh还能够在注册后执行远程系统中的单个命令后立即返回。具体的用法是在ssh命令后面附加一条命令，命令前后使用双引号括起来。在下面的例子中，假定当前系统中的用户想要了解远程系统iscas的Linux内核版本号，因而需要在远程系统中运行“uname -r”命令：

```
$ ssh iscas "uname -r"
gqxing@iscas's password:
2.6.31-16-generic
$
```

这种一次性地注册，执行远程命令，然后立即返回的功能是非常有用的。结合使用后面将要介绍的无密码注册功能，无论何时都能非常容易地获取远程系统的各种状态信息。

18.1.5 使用SCP替代FTP

从网络通信的角度来看，FTP的数据传输方式是不安全的，这是因为FTP协议在网络中传输的用户名、密码和数据没有采取任何加密措施。比较安全的方法是采用OpenSSH的SFTP

（Secure FTP）和SCP（Secure Copy）。

scp是OpenSSH中另外一个重要客户端软件，能够在不同的系统之间复制文件，其语法格式类似于常规的cp命令，简写如下：

```
scp [-P port] [[user@]host1:]file1 [[user@]host2:]file2
```

其中，“-P”选项用于指定TCP端口。第一个参数是源文件，第二个参数是目的文件。当需要复制远程系统中的文件时，SCP首先需要成功地注册到远程系统中，然后才能开始传输文件，因此要求提供远程系统的名字、用户名和密码。在引用远程系统中的文件名时，文件名前面应加用户名与系统名前缀（两者之间需要插入一个“@”字符，之后再加一个冒号“:”）。远程文件名或目录名的表示形式如下：

```
username@servername:filename
username@servername:directoryname
```

其中，servername既可以是远程系统的主机名，也可以是IP地址。另外，除非引用的是用户主目录中的文件，指定文件时应给出绝对路径名或针对主目录的相对路径名。例如，如果远程系统的IP地址为172.16.3.40，为了访问其中的/etc/profile文件，可以使用“172.16.3.40:/etc/profile”的形式表示远程系统中的文件。为了访问远程系统guest用户主目录中的.profile文件，可以使用“guest@iscas:.profile”的形式表示其中的文件。

1. 利用scp命令下载文件

例如，为了把远程系统中的/etc/profile文件复制到本地系统的指定目录/tmp中，可以使用下列命令：

```
$ scp gqxing@iscas:/etc/profile /tmp
gqxing@iscas's password:
profile                                100%  497    0.5KB/s   00:00
$
```

2. 利用scp命令上传文件

反之，把本地Linux系统中的文件复制到远程系统中也是一样的。例如，为了把本地系统中的/etc/hosts文件复制到远程的/tmp目录中，可以使用下列命令：

```
$ scp /etc/hosts gqxing@iscas:/tmp
gqxing@iscas's password:
hosts                                100%  368    0.4KB/s   00:00
$
```

18.1.6 使用SFTP替代FTP

OpenSSH还提供了一个SFTP程序。SFTP采用加密的SSH会话，实现FTP的文件传输功能。由于SFTP也具有FTP的目录浏览功能，故SFTP比SCP更方便实用，尤其是在不知道欲复制文件的确切位置时。但请注意，SFTP并不支持传统FTP的匿名注册功能。sftp的语法格式简写如下：

```
sftp [-b batchfile] host
sftp [[user@]host[:file [file]]]
sftp -b batchfile [user@]host
```

其中,“-b”选项用于指定一个批处理文件,其中包含get、put、cd或lcd等sftp子命令,以便实现非交互式的文件传输。第二种命令形式用于实现系统之间的简单文件复制。

下面是一个使用SFTP注册、查询文件,然后下载文件的例子:

```
$ sftp iscas
Connecting to iscas...
gqxing@iscas's password:
sftp> ls -l
drwxr-xr-x  2 gqxing  gqxing      4096 Nov 12 16:16 conf
drwxr-xr-x  2 gqxing  gqxing      4096 Nov 12 16:56 docs
-rw-r--r--  2 gqxing  gqxing       167 Nov 12 16:10 examples_desktop
-rw-r--r--  2 gqxing  gqxing    61090 Nov 12 16:10 filelist
drwxr-xr-x  2 gqxing  gqxing      4096 Nov 12 16:16 incl
drwxr-xr-x  2 gqxing  gqxing      4096 Nov 10 18:20 script
drwxr-xr-x  2 gqxing  gqxing      4096 Nov 11 19:16 src
.....
sftp> lcd /tmp
sftp> get filelist
Fetching /home/gqxing/filelist to filelist
/home/gqxing/filelist          100%   60KB  59.7KB/s   00:00
sftp> exit
$
```

18.1.7 SSH与SCP的无密码注册

常规情况下,每次使用ssh命令注册或使用scp命令复制远程系统的文件时,都需要提供密码,然后才能做进一步的处理。为了省略密码输入步骤,可以采用Shell脚本实现,但这需要把手动输入的密码(也即明码)放在脚本文件中。

实际上,适当地利用密钥配置文件,OpenSSH能够省略ssh远程注册与scp文件复制操作过程中的密码验证环节。为了实现无密码注册,客户端应首先建立OpenSSH连接,然后自动向服务器发送其密钥(公钥)。之后,服务器即可根据相应用户主目录中预定义的密钥列表,对收到的密钥进行比较。如果存在匹配的密钥,服务器将会允许ssh或scp自动注册。

用于远程系统注册与数据传输的密钥文件需要事先单独生成,生成后的密钥文件存储在服务器用户主目录的~/.ssh子目录中。其中,私钥和公钥分别存储在id_dsa和id_dsa.pub文件中,authorized_keys文件则用于存储所有授权的远程客户系统的公钥,使得远程客户系统能够以此用户身份注册到本地系统而无需提供密码。

按照上述方法实现无密码注册,唯一的限制是需要把密钥绑定到两个系统的IP地址上。之后,服务器即可使用预安装的密钥,对每一次的远程注册和文件传输进行验证。但这一功能特性并不适合于利用DHCP协议动态分配IP地址,致使IP地址经常发生变化的情况。

由于只需知道用户名,即可实现远程系统注册与文件传输,而无需提供密码。因此,这种做法存在一定的安全风险。在具体实现时,服务器与客户端之间最好采用普通用户账号,以免造成较大的破坏。

下面以gqxing用户为例,详细说明在实现无密码注册之前,OpenSSH的客户端与服务器双方事先都需要做哪些准备工作。

1. OpenSSH客户端配置

为了实现无密码的系统注册，OpenSSH客户端需要完成下列准备工作。

首先，在客户端与服务器系统中分别创建一个同名的gqxing用户，然后以gqxing用户的身份注册到客户端系统，使用ssh-keygen命令生成一对密钥。当系统提示输入一个与密钥相关联的密码时，直接按下Enter键。

```
$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/gqxing/.ssh/id_dsa): <Enter>
Enter passphrase (empty for no passphrase): <Enter>
Enter same passphrase again: <Enter>
Your identification has been saved in /home/gqxing/.ssh/id_dsa.
Your public key has been saved in /home/gqxing/.ssh/id_dsa.pub.
The key fingerprint is:
4c:57:c4:5e:b5:41:af:12:25:1b:ad:8f:16:98:bf:d5 gqxing@sinosoft
The key's randomart image is:
.....
$
```

其次，验证上述步骤生成的两个密钥文件是否均存储在gqxing用户主目录的ssh子目录中。其中，id_dsa文件中存储的是私钥，id_dsa.pub文件中存储的是公钥（用于提交远程服务器，解密客户端传输的加密数据）。

```
$ ls -l .ssh
total 12
-rw----- 1 gqxing gqxing 668 Dec 11 12:51 id_dsa
-rw-r--r-- 1 gqxing gqxing 605 Dec 11 12:51 id_dsa.pub
-rw-r--r-- 1 gqxing gqxing 884 Nov 21 11:01 known_hosts
$
```

最后，采用下列命令，把生成的公钥文件id_dsa.pub复制到远程系统的gqxing用户主目录中：

```
$ cd .ssh
$ scp id_dsa.pub gqxing@iscas:pub_key
gqxing@iscas's password:
id_dsa.pub                                100% 605      0.6KB/s   00:00
$
```

2. OpenSSH服务器配置

在OpenSSH服务器中，以gqxing用户的身份注册到系统，然后使用cat命令和“>>”重定向符号，把pub_key文件中的数据附加到authorized_keys文件的后面：

```
$ cat ~/pub_key >> .ssh/authorized_keys
$ rm ~/pub_key
$
```

authorized_keys文件包含所有OpenSSH客户端系统的公钥列表，如果表中列举的客户系统中的gqxing用户仍以同一用户身份连接到服务器，则无需提供密码。

3. 示例

在完成上述的全部设置之后，当OpenSSH客户端的gqxing用户仍以gqxing用户的身份，使用ssh、scp或sftp命令连接到远程主机，就不会再提示用户输入密码了。例如：

```
gqxing@sinosoft:~$ ssh iscas
.....
Last login: Fri Dec 11 20:28:34 2009 from sinosoft
gqxing@iscas:~$
```

下面是一个利用scp命令复制文件的例子，其情况同ssh命令一样，远程服务器也不要求用户提供密码：

```
gqxing@sinosoft:~$ scp iscas:/etc/hosts .
hosts                                                    100% 368      0.4KB/s   00:00
gqxing@sinosoft:~$
```

但这一方法对客户端的其他用户（包括超级用户）无效，即使他们也以gqxing用户的身份注册到远程服务器，仍需提供密码，示例如下：

```
cathy@sinosoft:~$ ssh -l qqxing iscas
qqxing@iscas's password:
.....
Last login: Fri Dec 11 23:20:10 2009 from sinosoft
qqxing@iscas:~$
```

18.1.8 OpenSSH的安全考虑

从先前的介绍和表1 8-1 中可知, Ubuntu Linux系统中的sshd守护进程通常都会监听服务器所有IP地址的22号TCP端口。为了安全起见, 可以把默认的22号端口改为自己约定的其他空闲端口号(如435等)。同时还需要把/etc/services文件中的下列端口定义:

```
ssh      22/tcp
ssh      22/udp
```

改为

```
ssh      435/tcp
ssh      435/udp
```

修改TCP端口之后，可在OpenSSH客户端中使用下列命令，注册远程系统：

```
$ ssh -p 435 iscas
gqxing@iscas's password:
.....
Last login: Fri Dec 11 23:50:10 2009 from sinosoft
$
```

为了使用修改后的435端口，把远程系统中的一个文件复制到本地系统的指定目录中，可以使用下列命令：

[illegible]

同样，为了使用TCP端口435，把本地系统中的/etc/hosts文件复制到远程的/tmp目录中，可以使用下列命令：

```
$ scp -P 435 /etc/hosts gqxing@iscas:/tmp
gqxing@iscas's password:
hosts                                100% 368    0.4KB/s   00:00
$
```

此外，还可以使用AllowUsers、AllowGroups、DenyGroups以及DenyUsers配置参数，或者这些参数的组合，限定用户或用户组的访问。例如，为了限定只有gqxing和cathy两个用户能够访问系统，可以在/etc/ssh/sshd_config配置文件中增加下列配置参数：

```
AllowUsers    gqxing cathy
```

重新启动sshd进程之后，除了gqxing与cathy两个用户之外，系统将会拒绝接受其他用户的注册，并输出拒绝访问的错误信息。示例如下：

```
$ ssh -l guest iscas
guest@iscas's password:
Permission denied, please try again.
guest@iscas's password:
Permission denied, please try again.
guest@iscas's password:
Permission denied (publickey,password).
$
```

修改OpenSSH配置文件之后，为使新的设置立即生效，需要重新启动sshd守护进程。示例如下：

```
$ sudo /etc/init.d/ssh restart
* Restarting OpenBSD Secure Shell server sshd          [ OK ]
$
```

18.2 Telnet远程注册

尽管OpenSSH可以取代Telnet，Telnet也确实存在安全方面的潜在问题，但并不是每一个系统都支持OpenSSH，如在Windows系统中，如果想用OpenSSH，需要从网上下载免费的开源软件，故Telnet仍然是应用比较广泛的通信手段之一。因此，这一节将简单介绍怎样在Linux系统中使用Telnet。

在Ubuntu Linux系统中，Telnet等传统网络应用通常分为两个软件包：一为客户端软件包（如telnet，其中含有telnet应用程序及其参考手册等）；一为服务器软件包（如telnetd，其中含有in.telnetd服务程序及其配置文件等）。新的Ubuntu Linux系统安装介质通常仅提供客户端软件包，因而不会安装网络服务器软件包。

由此可见，Linux系统中的用户通常只能利用Telnet注册到其他系统（如UNIX系统），不能注册到初始安装的任何Ubuntu Linux系统，甚至也不能使用Telnet注册到自己的Ubuntu Linux系统中。

如同第15章“系统启动与关机”所述，大部分系统守护进程（包括sshd等网络守护进程）通常都是由/etc/init.d目录中的Shell脚本直接控制，随着系统的启动而运行，随着系统的关机而

停止的。但是，传统的网络守护进程是由inetd统一控制的，而且按照实际的需要和请求才会启动。inetd网络总控进程是由/etc/inetd目录中的Shell脚本openbsd-inetd控制的，而且也会随着系统的启动而自动运行。

在系统启动之后的运行过程中，为了查询inetd当前的运行状态，可以使用下列命令（参见第10章“进程管理”）：

```
$ pidof inetd
2660
$
```

设置Telnet服务器

由于Ubuntu Linux系统的安装程序不会自动安装telnetd服务器软件包，为了支持Telnet远程注册访问，可以使用apt-get、aptitude或synaptic等软件工具安装telnetd服务器软件包：

```
$ sudo aptitude install telnetd
```

在安装telnetd软件包的过程中，系统将会把下列内容自动写入/etc/inetd.conf配置文件，同时运行/etc/inetd/openbsd-inetd脚本，启动inetd守护进程

```
telnet stream tcp nowait root /usr/sbin/tcpd /usr/sbin/in.telnetd
```

此后，只要拥有注册的用户账号，任何人都可以利用telnet命令注册，访问Linux主机系统。telnet命令的语法格式简写如下：

```
telnet [options] host [port]
```

其中，host是远程系统的主机名或IP地址，port是TCP端口号。当使用telnet命令连接远程主机时，系统将会显示一系列信息，提示用户输入用户名与密码。如果输入的用户名和密码通过验证，则可进入远程系统，打开默认的注册Shell，输出命令提示符。至此，用户即可输入命令，交互访问远程系统。如果用户名和密码不匹配，系统就会拒绝用户注册。

下面是一个简单的注册访问过程。其中，sinosoft系统中的用户以普通用户gqxing的身份注册到远程系统iscas，执行一个uname命令后立即退出远程系统：

```
$ telnet iscas
Trying 169.254.78.100...
Connected to iscas.
Escape character is '^]'.
Ubuntu 9.10
iscas login: gqxing
Password:
Last login: Fri Dec 11 21:17:24 CST 2009 from sinosoft on pts/0
Linux iscas 2.6.31-16-generic #52-Ubuntu SMP Thu Dec 3 22:00:22 UTC 2009 i686
.....
$ uname -n
iscas
$ exit
Connection closed by foreign host.
$
```

但是，这种注册访问仅限于普通用户账号，即只能使用普通用户账号注册到Linux系统，不能以超级用户root的身份直接注册。为了直接使用root注册（通常不建议这样做），尚需做进一步的设置。

通常，即使Ubuntu Linux系统允许超级用户在控制台上注册，也不允许任何人在控制台之外的其他终端上，以超级用户的身份注册到系统。这一限制实际上是由/etc/securetty文件控制的。securetty文件包含一系列设备名（其中省略了“/dev/”前缀），只有在其中指定的设备上使用root注册才是可以接受的。默认的/etc/securetty文件内容如下，其中列举的所有设备名实际上都是系统控制台及其别名：

```
$ cat /etc/securetty
console
.....
tty1
tty2
.....
$
```

当使用telnet命令注册时，不管是在同一个系统中，还是在不同的Linux系统之间，其设备名通常均为pts/n，其中的n是0、1、2…中的一个数字。为了突破上述限制，直接使用超级用户root注册，可在securetty文件中增加下列内容：

```
pts/0
pts/1
pts/2
.....
```

之后，即可使用root，以超级用户的身份注册到远程系统（假定超级用户root的账号已经解除封锁）：

```
$ telnet iscas
Trying 169.254.78.100...
Connected to iscas.
Escape character is '^]'.
Ubuntu 9.10
iscas login: root
Password:
Last login: Fri Dec 11 21:46:12 CST 2009 from sinosoft on pts/0
Linux iscas 2.6.31-16-generic #52-Ubuntu SMP Thu Dec 3 22:00:22 UTC 2009 i686
.....
$
```

18.3 FTP文件传输

FTP（File Transfer Protocol）是最常用的一种文件传输工具。作为一个基本的组成部分，大多数操作系统均提供完整的FTP网络应用，Linux系统也不例外。Ubuntu Linux系统采用安全的FTP守护进程vsftpd，支持FTP服务器功能。

18.3.1 设置vsftpd

同Telnet网络应用一样, 初始安装的Ubuntu Linux基本系统总是包含FTP客户端软件ftp, 但不包括服务器软件包vsftpd。为了提供FTP服务器功能, 需要使用apt-get、aptitude或synaptic等软件工具, 安装vsftpd服务器软件包:

```
$ sudo aptitude install vsftpd
```

一旦安装vsftpd服务器软件包之后, /etc/init.d目录中将会增加一个vsftpd脚本文件, 并会在系统的启动过程中自动运行vsftpd守护进程。

为了查询vsftpd守护进程当前的运行状态, 可以使用下列pidof命令获取指定进程的PID(参见第10章“进程管理”):

```
$ pidof vsftpd
2698
$
```

也可以使用netstat命令, 查询系统当前正在监听的所有TCP和UDP端口, 检查其中是否存在ftp。如果vsftpd守护进程没有运行, 下列netstat命令不会产生任何输出信息:

```
$ netstat -a | grep ftp
tcp        0      0  *:ftp          *:*            LISTEN
$
```

vsftpd守护进程采用/etc/vsftpd.conf配置文件规范其处理动作, 其中包含大量的运行时配置信息, 及以注释符号“#”为起始字符的注释行与配置示例, 注释行提供了许多有用的解释。必要时, 只需删除配置示例前的注释符号即可激活相应的配置变量。在修改vsftpd服务器的配置文件之后, 为使修改后的设置立即生效, 需要重新启动vsftpd守护进程。为此, 可以使用下列命令:

```
$ sudo /etc/init.d/vsftpd restart
* Stopping FTP server: vsftpd          [ OK ]
* Starting FTP server: vsftpd          [ OK ]
$
```

通常, vsftpd守护进程拒绝超级用户root访问FTP服务器。为了取消这一限制, 可以编辑/etc/ftpusers文件, 在root之前插入一个注释符号“#”。

18.3.2 vsftpd.conf配置文件

在开始运行之初, vsftpd守护进程首先会读取配置文件, 按照配置文件中的参数设置采取相应的处理动作。vsftpd的默认配置文件为/etc/vsftpd.conf, 必要时也可以在vsftpd的命令行参数中指定其他配置文件。

vsftpd.conf配置文件的语法非常简单, 每一行或者是一个注释行, 或者是一个参数设置。以注释符号“#”为起始字符的注释行将被忽略。vsftpd.conf配置文件参数设置的语法格式如下:

```
parameter= value
```

**注意**

在设置配置参数时，等号“=”前后不能加任何空格字符。vsftpd.conf配置文件中的每个配置参数都有一个默认的设置。如有必要，可以做适当的修改（在修改任何配置参数之后，不要忘记重新启动/etc/init.d/vsftpd脚本）。

在使用或修改vsftpd的配置文件中，首先需要了解vsftpd守护进程的默认处理方式。下面是其中的一部分：

- vsftpd支持FTP匿名访问。如果想要禁止匿名访问，可把anonymous_enable参数设置为NO。
- 若想利用服务器系统中的用户账号实名访问FTP服务器，需要删除local_enable参数前面的注释符号“#”。
- vsftpd只允许FTP匿名用户下载文件，但不能上传文件。如果允许上传文件，需要把anon_upload_enable参数设置为YES（同时还需要把write_enable参数设置为YES）。此外还需要使用下列命令，修改FTP匿名用户主目录/home/ftp中pub共享目录的访问权限。否则，/home/ftp与/home/ftp/pub目录的默认访问权限不允许其他用户创建文件，因而无法上传文件。

```
# chmod 777 /home/ftp/pub
```

- vsftpd禁止匿名用户在FTP服务器中创建子目录。如果允许匿名用户创建子目录，需要删除anon_mkdir_write_enable参数前面的注释符号“#”（同时还需要把write_enable参数设置为YES）。同样，还要使用下列命令修改FTP匿名用户主目录/home/ftp中pub共享目录的访问权限。否则，/home/ftp和/home/ftp/pub目录的访问权限不允许其他用户创建子目录。

```
# chmod 777 /home/ftp/pub
```

- vsftpd采用/var/log/vsftpd.log文件作为FTP服务器默认的日志文件。如有必要，也可以利用xferlog_file配置参数，改换为其他文件。
- FTP匿名用户的主目录为/home/ftp。如想改用其他目录，需要修改anon_root的参数设置。FTP匿名访问总是具有一定的风险，如果采用默认值，且允许上传文件，用户能够随心所欲地把文件写到FTP公共目录中，从而把整个/home文件系统分区耗费殆尽。因此，最好的办法是把FTP匿名用户主目录改换到一个专用的文件系统分区中。

有关vsftpd.conf配置文件参数如何设置的更多说明，参见表18-2。

表18-2 vsftpd.conf配置文件的部分配置参数

配置参数	默认值	简单说明
anon_mkdir_write_enable	N C	指定是否允许匿名用户创建新的子目录。如果设为YES，意味着允许匿名用户在FTP匿名用户主目录的共享子目录pub中创建新目录。启用这个功能特性要求同时启用write_enable配置参数，且FTP匿名用户还应具有写共享子目录pub的访问权限
anon_other_write_enable	N C	如果设为YES，意味着允许匿名用户执行除上传文件和创建目录之外的其他写操作，如删除文件、重新命名文件等。通常不建议启用这个功能特性

(续表)

配置参数	默认值	简单说明
anon_upload_enable	N C	指定是否允许FTP匿名用户上传文件。如果设为YES, 意味着允许匿名用户上传文件。启用这个功能特性要求同时启用write_enable配置参数, 且FTP匿名用户还应具有写上传目录(如pub)的访问权限
anon_world_readable_only	YES	如果设置为YES, 匿名用户只能下载任何用户均可读的文件
anonymous_enable	YES	指定是否允许匿名注册访问。如果启用, 意味着可以使用用户名anonymous或ftp匿名注册
ascii_download_enable	N C	启用时, 表示允许采用ASCII模式下载数据文件。通常均采用二进制模式传输数据
ascii_upload_enable	N C	启用时, 表示允许采用ASCII模式上传数据文件。通常均采用二进制模式传输数据
background	N C	启用时, vsftpd守护进程将会在启动后进入“监听”模式, 以后台方式运行。也就是说, 在vsftpd开始运行之后, 其控制权将会立即转交给Shell
dir_list_enable	YES	指定是否允许用户使用目录文件的列表显示命令(如dir或ls等)。如果设为NO, 表示拒绝用户使用任何目录文件的列表显示命令
download_enable	YES	指定是否允许用户下载文件。如果设为NO, 表示拒绝执行任何文件下载请求
force_dot_files	N C	指定是否在列表显示目录文件时总是显示以句点“.”为起始字符的隐藏文件和目录, 而不管是否采用“-a”选项。注意, 这个配置参数不影响“.”和“..”两个特殊目录项的显示规则
guest_enable	N C	如果设为YES, vsftpd将会把所有非匿名用户看做guest用户。在Ubuntu Linux系统中, guest用户通常映射为guest_username配置参数指定的用户, 即ftp
hide_ids	N C	如果设置为YES, 在显示目录文件列表时, 其用户与用户组均显示为ftp
listen	NO (实为YES)	启用时, vsftpd可以采用独立运行模式, 这意味着无需由inetd控制, 可以直接运行vsftpd, 由vsftpd自己负责监听和处理FTP访问服务
local_enable	N C	控制是否允许本地系统中的注册用户访问FTP服务器。如果设为YES, 意味着可以使用服务器系统/etc/passwd文件中的普通用户账号访问FTP服务器。如果允许以实名方式访问FTP服务器, 这个参数必须设为YES
log_ftp_protocol	N C	启用时, vsftpd将会采用标准的xferlog格式, 记录所有的FTP请求与响应等日志信息。这个配置参数主要用于调试的目的
ls_recurse_enable	N C	指定是否允许递归地列出目录中的所有文件。如果设为YES, 表示允许用户使用“ls -R”命令
no_anon_password	N C	指定匿名注册时是否提示用户输入密码。如果设为YES, vsftpd不会提示用户输入匿名密码, 这意味着输入ftp或anonymous匿名用户名之后将会直接注册到FTP服务器

(续表)

配置参数	默认值	简单说明
syslog_enable	NC	如果设为YES, 通常写入/var/log/vsftpd.log日志文件的任何日志信息将会转而写入系统日志文件中
tilde_user_enable	NC	如果设为YES, vsftpd将会尝试解析“~username/dir file”形式的路径名。注意, vsftpd通常总是解析“~”和“~/something”形式的路径名, 其中波浪号“~”表示用户的主目录
use_localtime	NC	如果设为YES, vsftpd将会以本地时区的时间显示目录文件列表。默认的时区为GMT
userlist_deny	YES	如果userlist_enable设为YES, 需要进一步考察userlist_deny配置参数。如果userlist_deny采用默认值YES, vsftpd将会拒绝vsftpd.user_list (参见userlist_file配置参数) 文件中列举的用户注册访问。如果设为NC, 则允许user_list文件中列举的用户注册访问, 而拒绝其他用户注册访问。在拒绝用户注册时, vsftpd将会直接输出拒绝信息, 而不会提示用户输入密码
userlist_enable	NC	如果设为YES, vsftpd守护进程将会从userlist_file配置参数指定的文件 (即/etc/vsftpd.user_list) 中加载用户名列表。如果试图使用其中列举的某个用户名注册, 将会立即遭到拒绝, 而且根本就不提供密码验证的机会。这样做是非常有用的, 可以防止明文密码的传输。参见userlist_deny配置参数
write_enable	NC	用于控制用户是否能够使用FIP与文件写操作有关的子命令, 其中包括: STOR、DELE、RNFR、RNTO、MKD、RMD、APPE和SITE。实际上, 这个配置参数主要用于间接控制用户是否能够上传文件, 在FIP服务器中创建目录, 以及删除文件等
xferlog_enable	NO (实为YES)	如果设为YES, vsftpd将会利用日志文件详细记录与文件上传和下载有关的信息。vsftpd守护进程默认的日志文件为/var/log/vsftpd.log, 但可以利用配置参数vsftpd_log_file指定一个新的日志文件
xferlog_std_format	NC	如果设为YES, vsftpd将会按照标准的xferlog格式把文件传输的日志信息记录到/var/log/xferlog日志文件中。如果设为NC, 其数据格式的可读性更强
data_connection_timeout	300 (实为120)	以秒为单位的超时值。这是在FIP数据传输期间vsftpd允许的最大停顿时间。如果出现超时, vsftpd将会断开用户的FIP连接
file_open_mode	0666	上传文件时赋予新建文件的访问权限。如果上传的文件大多为可执行程序, 应把这个参数设置为0777
idle_session_timeout	300 (实为600)	以秒为单位的超时值。这是vsftpd在用户执行FIP命令期间允许的最大空闲时间。如果出现超时, vsftpd将会断开用户的FIP连接
listen_port	21	如果采用独立运行模式, vsftpd守护进程将会采用这个参数定义的端口监听用户的FIP连接请求
max_clients	0 (没有限制)	指定vsftpd能够接受的最大FIP连接数量。超出此限的FIP连接请求将会收到一个错误信息
max_login_fails	3	指定最大的注册尝试次数。如果注册失败超过指定的次数, FIP将会终止注册会话

(续表)

配置参数	默认值	简单说明
anon_root	/home/ftp	用于指定匿名用户的主目录。在匿名用户注册成功之后, vsftpd将会使用这个目录作为初始工作目录
banner_file	无	用于指定一个文件, 其中包含用户注册到FTP服务器后的显示信息。如果设定了此配置参数, 指定文件中的内容将会取代ftp_banner配置参数提供的字符串信息
deny_file	无	这个配置参数用于设置文件或目录的名字模式, 表示不允许访问匹配指定模式的目录或文件, 如拒绝下载匹配的文件、拒绝改换到匹配的目录以及拒绝访问其中的文件等。注意, vsftpd能够处理的正则表达式只是一个简单的实现, 或者说只是正则表达式的一个子集。因此, 设置后需要仔细测试。此外, 建议尽可能采用文件系统的访问权限设置。vsftpd能够识别星号“*”、问号“?”和花括号“{}”, 而且这里的匹配检查仅限于路径名的文件名部分, 如a/b/?, 但不支持a/?/c。例如: deny_file=*.mp3,*.mov表示拒绝访问其扩展名为.mp3和.mov的任何文件
ftp_username	ftp	指定除anonymous之外, 可以匿名访问FTP服务器的其他匿名用户名, 其主目录也是FTP匿名用户能够访问的根目录
ftpd_banner	(vsFTPd 2.0.7)	用于定义一个字符串, 如“Welcome to FTP service”, 作为建立FTP连接后的欢迎信息, 以替代vsftpd提供的默认信息“(vsFTPd 2.0.7)”, 其中2.0.7是vsftpd当前的版本号
guest_username	ftp	用于定义guest用户映射的真实用户, 以统一规范非匿名用户的访问权限
hide_file	无	这个配置参数用于设置文件或目录的名字模式, 表示在显示目录文件列表时隐藏匹配指定模式的目录或文件。其注意事项参见deny_file配置参数的说明
listen_address	none	如果vsftpd处于独立运行模式, 可以使用这个配置参数指定一个IP地址, 强制vsftpd监听指定的地址(通常, vsftpd将会监听所有的本地网络接口)
local_root	\$HOME(用户主目录)	这个配置参数用于确定在非匿名的本地用户注册之后, vsftpd将引导用户进入的工作目录
userlist_file	/etc/vsftpduser_list	如果userlist_enable设为YES, vsftpd将会使用userlist_file设定的文件作为用户访问控制文件的名字, 加载其中的用户列表
vsftpd_log_file	/var/log/vsftpdlog	指定vsftpd常规日志文件的名字。注意, 仅当配置参数xferlog_enable设为YES, xferlog_std_format设为NO时, 才会使用这个参数指定的文件记录vsftpd的常规日志信息。如果syslog_enable配置参数设为YES, 日志信息将会写到系统日志文件, 而非vsftpd自己的日志文件
xferlog_file	/var/log/vsftpdlog	指定文件传输日志文件的名字。注意, 仅当配置参数xferlog_enable设为YES之后, vsftpd才会使用这个文件记录文件传输的日志信息

18.3.3 ftp命令

在完成上述设置, 或适当地修改配置文件之后, 即可开始利用ftp命令访问FTP服务器, 传输文件。ftp命令的语法格式简写如下:

```
ftp [-debugin] [host [port]]
```

其中，“-i”选项表示在传输多个文件期间，关闭交互提示与确认模式。“-n”选项禁止使用自动注册功能建立连接。如果未禁止自动注册，ftp将会检测用户主目录中的netrc文件，检索其中提供的远程系统账号信息。如果检索失败，或netrc文件的访问权限设置不正确，ftp将会提示用户输入远程系统的用户名或密码。host是FTP服务器的主机名或IP地址。port是ftp服务的端口号。

ftp还提供了许多内部子命令，借助于这些内部子命令，可以实现各种文件传输功能。为了查询ftp提供了哪些可用的内部子命令，可在ftp命令提示符“ftp>”下输入“?”命令，然后按“Enter”键。表18-3给出了部分常用的ftp内部子命令及简单说明。

表18-3 常用的ftp内部子命令

ftp内部子命令	简单说明
! <i>cmd</i>	在本地系统的Shell环境中执行指定的命令。如果未提供命令，则进入交互式的Shell运行环境
ascii	以ASCII代码形式传输文件。这是默认的传输方式
binary	以二进制代码形式传输文件
bye	终止与远程FTP服务器的会话，并退出ftp命令
cd <i>remote-dir</i>	把远程系统上的当前工作目录改换到指定的目录
delete <i>remote-file</i>	删除远程系统中的指定文件
dir [<i>remote-dir</i>] [<i>local-file</i>]	列出远程主机指定目录下的文件和目录，并把输出结果保存在指定的本地文件中。如果未指定远程目录，则默认为远程主机的当前工作目录。如果未指定本地文件，或指定的本地文件为“-”，则输出结果将直接送到终端显示
get <i>remote-file</i> [<i>local-file</i>]	下载远程文件，并储存在本地系统。如果未指定本地文件名，则采用与远程文件相同的名字命名本地文件
help [<i>cmd</i>]	显示指定内部命令的说明信息。如果未指定任何命令，ftp将会列出所有可用的内部子命令
lcd [<i>local-dir</i>]	改变本地系统上的当前工作目录。如果未指定目录，则使用用户的主目录
ls [-a] [<i>remote-dir</i>] [<i>local-file</i>]	以长列表的形式，列出FTP服务器中指定目录或当前工作目录下的文件或子目录，给出文件的类型与访问权限、链接数、文件属主、用户组、文件的字节数及最后一次修改时间等属性，其效果类似于Linux系统中的“ls -l”命令。其中，“-a”选项意味着列出所有文件，包括以句点“.”为首字符的隐藏文件。如果未指定远程目录，默认值为当前工作目录。如果未指定本地文件名，或者本地文件名是“-”，输出内容将直接送到终端
mget <i>remote-files</i>	按文件名生成机制展开指定的远程文件名，对每个文件执行一次get操作。下载的文件将存储在本地工作目录或之前由“lcd dir”命令指定的目录中。如果需要，也可以首先使用“! mkdir dir”命令创建新的本地目录
mkdir <i>remote-dir</i>	在远程系统上创建指定的目录
mput <i>local-files</i>	一次上传多个文件。其具体实现是展开由通配符表示的本地文件名，使之作为put命令的参数，对每个文件执行一次put操作
put <i>local-file</i> [<i>remote-file</i>]	把本地文件上传到FTP服务器。如果未指定远程文件名，则使用本地文件名作为远程文件名

(续表)

ftp内部子命令	简单说明
pwd	显示FTP服务器的当前工作目录
quit	同bye

例如, 为了访问FTP服务器iscas, 可以使用下列命令:

```
$ ftp iscas
Connected to iscas.
220 (vsFTPd 2.2.0)
Name (iscas:gqxing):
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

18.3.4 FTP应用

在传输文件时, 可以使用ftp的get子命令下载文件, 使用put子命令上传文件。为了查询FTP服务器当前目录中存在的文件, 然后下载其中的某个文件, 可以使用ls或dir等子命令。使用cd命令可以进入其他目录。为了退出FTP服务器, 可以使用quit或bye等子命令。例如:

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x  2 1000    1000      4096 Nov 12 16:16 conf
drwxr-xr-x  2 1000    1000      4096 Nov 12 16:56 docs
-rw-r--r--  2 1000    1000       167 Nov 12 16:10 examples_desktop
-rw-r--r--  2 1000    1000    61090 Nov 12 16:10 filelist
drwxr-xr-x  2 1000    1000      4096 Nov 12 16:16 incl
drwxr-xr-x  2 1000    1000      4096 Nov 10 18:20 script
drwxr-xr-x  2 1000    1000      4096 Nov 11 19:16 src
.....
226 Directory send OK.
ftp> lcd /tmp
local directory now /tmp
ftp> get filelist
local: filelist remote: filelist
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for filelist (61090 bytes).
226 File send OK.
61090 bytes received in 0.01 secs (11107.5 kB/s)
ftp>
```

当需要一次传输多个文件时, 可以使用ftp的mget或mput子命令。但在使用mget或mput子命令时, ftp通常总是在传输每一个文件之前提请用户确认一次。如果文件太多, 这一确认方式将不胜其烦。为了避免这一麻烦, 可以使用ftp的“-i”选项, 关闭提示与确认模式, 示例如下:


```

$ ftp -i iscas
Connected to iscas.
220 (vsFTPD 2.2.0)
Name (iscas:gqxing):
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd /usr/include/sys
250 Directory successfully changed.
ftp> lcd tmp
Local directory now /tmp
ftp> mget *.h
.....
local: wait.h remote: wait.h
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for wait.h (6472 bytes).
226 File send OK.
6472 bytes received in 0.00 secs (4945.5 kB/s)
local: xattr.h remote: xattr.h
200 PORT command successful. Consider using PASV.
150 Opening BINARY mode data connection for xattr.h (4337 bytes).
226 File send OK.
4337 bytes received in 0.00 secs (4108.0 kB/s)
ftp> quit
221 Goodbye.
$

```

18.3.5 FTP自动注册

每次使用ftp命令传输文件之前，都需要提供用户名和密码，才能建立与FTP服务器之间的连接。为了启用FTP的自动注册功能，简化操作步骤，用户可以在自己的主目录中创建一个netrc文件，把远程主机名、用户名和密码等信息加到其中。netrc文件的语法格式如下：

```
machine rhostname login username [password password]
```

其中，关键字machine用于定义远程系统的主机名。login用于指定远程系统中的有效用户名。password用于提供明文形式的密码（为了安全起见，这个字段可以省略）。一旦设置了这个文件，即可实现FTP服务器的自动注册。

例如，为了使某个系统中的用户gqxing能够自动注册到beijing、iscas和sinosoft三个FTP服务器，可以创建下列netrc文件（假定gqxing在这三个FTP服务器中的注册用户名也是gqxing）：

```

$ cat $HOME/.netrc
# $HOME/.netrc file
#
machine beijing login gqxing password hereiam
machine iscas login gqxing password itsme
machine sinosoft login gqxing password helloagain
$

```




由于.netrc文件中可能包含密码,且密码是以明文形式给出的,FTP要求必须使用下列命令设置.netrc文件的访问权限,确保只有用户自己能够读写此文件。否则,FTP将会拒绝执行自动注册过程。

```
$ chmod 400 $HOME/.netrc &@*Úchmod 600 $HOME/.netrc&©
$
```

在完成上述两个步骤之后,只要输入ftp命令,立刻就会建立FTP连接,进入ftp命令状态,从而省略输入用户名和密码两个步骤。示例如下:

```
$ ftp iscas
Connected to iscas.
220 (vsFTPD 2.2.0)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

18.3.6 FTP安全考虑

FTP具有一定的安全缺陷,但可以采取必要的措施加以克服。例如,可以限制某些用户实名访问FTP服务器等。但需要注意的是,在用户注册和数据传输过程中,FTP没有采取任何加密措施。

1./etc/ftpusers文件

为了实现FTP的访问控制,可以采用/etc/ftpusers文件,限制指定的用户访问FTP服务器。在安装vsftpd软件包之后,其中提供的ftpusers文件通常包含超级用户root,以及daemon、bin与sys等系统用户。

为了限制其他用户访问FTP服务器,可以把相应的用户名加到ftpusers文件中。其中,每个用户名占用一行。如果允许用户访问FTP服务器,可在用户名前增加注释符号“#”,或从ftpusers文件中直接删除相应的用户名。

下面是一个ftpusers文件的例子。其中,由于用户名root之前增加了一个注释符号“#”,因而允许超级用户root访问FTP服务器,但禁止guest等用户访问。

```
# cat /etc/ftpusers
# /etc/ftpusers: list of users disabled FTP access. See ftpusers(5).

# root
.....
guest
#
```

在数据传输期间,FTP不会加密用户提供的密码,因而有可能增加数据或密码在网上传输的风险。最好的做法是继续保持ftpusers文件中原有的用户设置,尤其不要为超级用户root开禁。为了安全起见,还可以再增加其他用户名。

2./etc/vsftpd.user_list文件

与ftpusers文件相比,vsftpd.user_list文件具有更灵活的访问控制措施。取决于vsftpd.conf配

置文件中`userlist_deny`参数的设置，可以拒绝`vsftpd.user_list`文件中列举的用户访问FTP服务器，也可以允许`vsftpd.user_list`文件中列举的用户访问FTP服务器。若采用`vsftpd.user_list`文件控制FTP服务器的访问，通常可以不考虑`ftpusers`文件。尤其是，如果采用`vsftpd.user_list`文件控制FTP服务器的访问，允许访问的用户名不应出现在`ftpusers`文件中。

至于是否采用`vsftpd.user_list`文件控制FTP服务器的访问，取决于`vsftpd.conf`配置文件中的两个参数设置：即`userlist_enable`与`userlist_deny`。`userlist_enable`配置参数的默认值为NO，表示不使用`vsftpd.user_list`文件。如果启用了`userlist_enable`配置参数，还需要进一步考察`userlist_deny`配置参数。`userlist_deny`参数的默认值为YES，表示FTP服务器会拒绝`vsftpd.user_list`文件中列举的用户访问；如果把`userlist_deny`设置为NO，意味着允许`vsftpd.user_list`文件中列举的用户访问FTP服务器，而拒绝其他用户访问。在拒绝用户访问FTP服务器时，`vsftpd`将直接输出拒绝信息，而不会提示用户输入密码。

即使安装了`vsftpd`软件包，Ubuntu Linux系统也不提供`vsftpd.user_list`文件，必要时可由网络管理员自行创建。如同`ftpusers`文件，`vsftpd.user_list`文件通常不应包含超级用户`root`，以及`bin`、`daemon`与`adm`等系统用户。根据需要，可以增加新的用户名，或删除（注解掉）其中的任何用户。

3. 匿名上传文件

如果匿名用户想把文件上传到FTP服务器，首先应修改`/home/ftp/pub`目录的访问权限，然后按照下列步骤，在`/home/ftp/pub`目录中创建一个具有写访问权限的共享子目录（这将允许用户上传自己的文件，但不能访问其他用户上传的文件）：

```
$ sudo chmod 777 /home/ftp/pub
$ sudo mkdir /home/ftp/pub/upload
$ sudo chmod 722 /home/ftp/pub/upload
$
```

4. 使用SCP替代FTP

FTP的一大缺点是采用明码方式传输数据，包括用户提交的用户名和密码，使用户的账号信息容易遭受非法侵袭和窃取。SCP（Secure Copy）与SFTP（Secure FTP）能够提供加密的数据传输功能，因而可用以替代FTP。但美中不足的是，SCP并不支持FTP的匿名用户访问功能。

参 考 文 献

- [1] William Von Hagen. Ubuntu Linux Bible[M]. Wiley Publishing, Inc., 2007.
- [2] Mark G Sobell. A Practical Guide to Ubuntu Linux[M]. Prentice Hall, 2008.
- [3] Mark G Sobell. A Practical Guide to Linux Commands, Editors, and Shell Programming[M]. Prentice Hall PTR, 2005.
- [4] Chris Negus. Linux Bible 2007 Edition[M]. Wiley Publishing, Inc., 2007.
- [5] Peter Harrison. Linux Quick Fix Notebook[M]. Prentice Hall PTR, 2005.
- [6] Kenneth Rosen. UNIX: The Complete Reference, Second Edition[M]. McGraw-Hill, 2007.
- [7] Ubuntu Linux Documentations (<https://help.ubuntu.com>)
- [8] Mendel Cooper. Advanced Bash-Scripting Guide (<http://download.csdn.net/source/949704>)